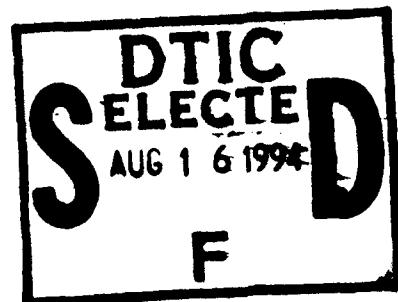


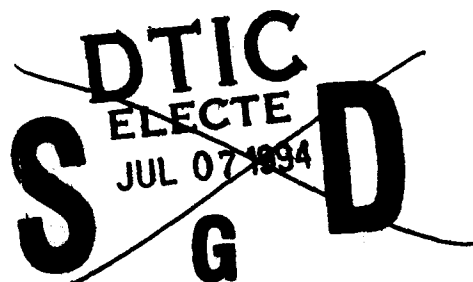
AD-A282 947



MSL-93



*Proceedings of the Second
International Workshop on
MULTISTRATEGY LEARNING*



*May 26 - 29, 1993
Harpers Ferry*

**Edited by Ryszard S. Michalski
and Gheorghe Tecuci**

DTIC QUALITY INSPECTED

**Sponsored by the
Office of Naval Research
and the
Center for Artificial Intelligence**

George Mason University

**Document has been approved
for release and distribution
this is confirmed**



Proceedings of the Second International Workshop on
MULTISTRATEGY LEARNING
(MSL-93)

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By <i>per lti</i>	
Dist. Option /	
Availability Codes	
Dist	Avail. and/or Special
A-1	

May 26-29, 1993
Harpers Ferry

DTIC QUALITY INSPECTED 2

Edited by Ryszard S. Michalski and Gheorghe Tecuci

N 00 014 -93 -1-0060

**Sponsored by the Office of Naval Research and
the Center for Artificial Intelligence, George Mason University**

94-25778



32976

94-629-162

Organizers

Ryszard S. Michalski, General Chairman
George Mason University

Gheorghe Tecuci, Program Chairman
George Mason University and Romanian Academy

Program Committee

Jaime Carbonell, *Carnegie Mellon University*
Yves Kodratoff, *CNRS & University of Paris-South*
Stan Matwin, *University of Ottawa*
Raymond Mooney, *University of Texas at Austin*
Katharina Morik, *University of Dortmund*
Michael Pazzani, *University of California at Irvine*
Luc de Raedt, *Catholic University of Leuven*
Ashwin Ram, *Georgia Institute of Technology*
Stuart Russell, *University of California at Berkeley*
Lorenza Saitta, *University of Torino*
Derek Sleeman, *University of Aberdeen*
Jude Shavlik, *University of Wisconsin*
Kurt VanLehn, *University of Pittsburgh*
Bradley Whitehall, *United Tech. R.C., East Hartford*
David Wilkins, *U. of Illinois at Urbana-Champaign*
Jianping Zhang, *Utah State University*

Local Arrangements

Michael Hieb	Nina Kaull	Janusz Wnek
<i>George Mason University</i>		

Auxiliary Reviewers

Jerzy Bala	Michael Hieb	Peter Pachowicz
Eric Bloedorn	David Hille	Bradley L. Richards
Mark Craven	Ibrahim Imam	Steven Salzberg
Steven Donoho	Carl Kadie	C. Tsatsarakis
David Duff	Ken Kaufman	Bradley Utz
Tomasz Dybala	Volker Klingspor	Haleh Vafaie
Pete Edwards	Ockkeun Lee	Steffo Weber
Matjaz Gams	Rich Maclin	Janusz Wnek
Hugo de Garis	Jeff Mahoney	John Zelle

Copyright ©1993, Center for Artificial Intelligence
George Mason University, 4400 University Dr, Fairfax, VA 22030-4444, USA
Email: airc@aic.gmu.edu, Tel: 703 993-1719, Fax: 703 993-3729

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means - electronic, mechanical, recording or otherwise, without the permission of the publisher.

ISBN 1-55860-305-0

Foreword

This volume contains the papers accepted for presentation at the Second International Workshop on Multistrategy Learning (briefly, MSL-93), held in Harpers Ferry, WV, May 26-29, 1993. The workshop was sponsored by the Office of Naval Research and organized by the Center for Artificial Intelligence at George Mason University.

The papers represent research on multistrategy learning conducted at leading research laboratories in the U.S. and other countries, such as Austria, Australia, Belgium, Germany, Italy, Japan, Romania, Slovenia, Spain, and United Kingdom. The presence of participants from so many countries is an indication of a truly international significance of the research in this area. To help the reader capture the variety of research directions, papers have been grouped into five categories, according to their primary theme: general issues, knowledge base refinement, cooperative integration, multiple computational strategies, special topics and applications.

Since multistrategy learning is one of the newest directions in the study and development of systems with learning capabilities, a brief explanation of its aims may be useful here. Research in this area concerns the development of learning systems that employ two or more inferential and/or computational strategies in a learning process. Though initial research had been primarily oriented toward integrating different inferential strategies (i.e., different types of inference), more recent research shows a trend to integrate also different computational strategies (i.e., different knowledge representations and associated processing methods). These Proceedings reflect this trend by including a section on methods for integrating such multiple computational strategies. Multistrategy learning systems are of increasing research interest due to their potentially significant advantages over monostrategy systems. Such systems can learn from a greater variety of inputs, with different amounts of prior knowledge, and generate different kinds of knowledge. Consequently, they could be useful for a wide range of practical problems. Since human learning is inherently multistrategy, the research in this area is also of significant importance to the study of human learning, and has opened new opportunities for a cross-fertilization of the two fields. Multistrategy learning workshops serve as a forum for researchers to present and discuss their recent research in this new, rapidly evolving and very challenging area.

We gratefully acknowledge the support from the Office of Naval Research, and express our special thanks to Lt. Comm. Robert Powell, without whose interest and encouragement this Workshop would not have happened.

We thank Dr. Su-Shing Chen and Dr. Andrew Sage who have honored the workshop with invited presentations.

We also thank the many individuals who helped in the organization and conduct of the workshop.

The Program Committee members and the auxiliary reviewers provided careful and timely reviews of the submitted papers. Their assistance was indispensable for insuring the high quality of the contributions.

Michael Hieb, Nina Kaull, and Janusz Wnek were in charge of the local organization. They diligently directed and executed many organizational aspects of the workshop.

The research assistants of the GMU Center for Artificial Intelligence, in particular, Jerzy Bala, Eric Bloedorn, Tomasz Dybala, Ibrahim Imam, Ken Kaufman, Mark Maloof, Alan Schultz, and Haleh Vafaie, provided invaluable help in handling many technical and managerial details. They are a great and reliable team, whose help cannot be overstated.

The Second International Workshop on Multistrategy Learning was the result of the enthusiastic work and the contribution of all the people mentioned above. We sincerely thank everyone for their help.

Ryszard S. Michalski and Gheorghe Tecuci

Contents

I. General Issues

Knowledge Representation for Multistrategy Task-adaptive Learning:.....	3
Dynamic Interlaced Hierarchies <i>Michael R. Hieb and Ryszard S. Michalski</i>	
Better Learners Use Analogical Problem Solving Sparingly.....	19
<i>Kurt VanLehn and Randolph M. Jones</i>	
Multistrategy Learning: An Analytical Approach.....	31
<i>Matija Drobnic and Matjaz Gams</i>	
Reflection and Analogy in Memory-based Learning.....	42
<i>Enric Plaza and Josep Lluís Arcos</i>	
Learning Scope, Task Analysis, and Sharable Components.....	50
<i>Erik M. Altmann</i>	
INTELOG: A Framework for Multistrategy Learning.....	58
<i>Wayne T. Jenkins</i>	

II. Knowledge Base Refinement

Symbolic Revision of Theories with M-of-N Rules.....	69
<i>Paul T. Baffes and Raymond J. Mooney</i>	
Knowledge Base Refinement Through a Supervised Validation of Plausible Reasoning.....	76
<i>Gheorghe Tecuci and David Duff</i>	
Learning to Survive.....	92
<i>Luc De Raedt, Erlend Bleken, Vincent Coget, Chaouat Ghil, Bart Swennen, Maurice Bruynooghe</i>	
MUSKRAT: A Multistrategy Knowledge Refinement and Acquisition Toolbox.....	107
<i>Nicolas Graner and Derek Sleeman</i>	

III. Cooperative Integration

Plausible Explanations and Instance-based Learning in Mixed Symbolic/Numeric Domains...	123
<i>Gerhard Widmer</i>	
k-DT: A Multi-Tree Learning Method.....	138
<i>David Heath, Simon Kasif and Steven Salzberg</i>	

Meta-Learning for Multistrategy and Parallel Learning	150
<i>Philip K. Chan and Salvatore J. Stolfo</i>	
Conceptual Clustering of Events Using Statistical Split Criteria.....	166
<i>Bradley L. Whitehall and David J. Sirag, Jr.</i>	
Cooperation of Data-driven and Model-based Induction Methods for Relational Learning	180
<i>Edgar Sommer</i>	
Multistrategy Constructive Induction: AQ17-MCI	188
<i>Eric Bloedorn, Janusz Wnek and Ryszard S. Michalski</i>	

IV. Multiple Computational Strategies

Extracting Symbolic Rules from Artificial Neural Networks	207
<i>Mark W. Craven and Jude W. Shavlik</i>	
A Multistrategy Learning Scheme for Assimilating Advice in Embedded Agents	218
<i>Diana F. Gordon and Devika Subramanian</i>	
REGAL: An Integrated System for Learning Relations Using Genetic Algorithms.....	234
<i>Atilio Giordana and Lorenza Saitta</i>	
Incremental Genetic Programming and Neural Net Learning: A Case Study	250
<i>Hugo de Garis</i>	

V. Special Topics and Applications

A Multistrategy Case-based and Reinforcement Learning Approach to.....	259
Self-improving Reactive Control Systems for Autonomous Robotic Navigation	
<i>Ashwin Ram and Juan Carlos Santamaria</i>	
A Machine Learning Approach to Document Understanding	276
<i>Florian Esposito, Donato Malerba, Giovanni Semeraro and Michael J. Pazzani</i>	
Towards GEST-3D: Learning Relations in 3-D Shapes	293
<i>Jakub Segen</i>	
Thinking and Seeing in Game Playing:.....	301
Integrating Pattern Recognition and Symbolic Learning	
<i>Susan L. Epstein and Jack Gelfand</i>	
Applying Multiple Learning Strategies for Classifying Public Health Data	309
<i>John W. Sheppard</i>	

Author Index

I. General Issues

Multitype Inference in Multistrategy Task-adaptive Learning: Dynamic Interlaced Hierarchies

Michael R. Hieb and Ryszard S. Michalski
Center for Artificial Intelligence
George Mason University, Fairfax, VA
hieb@aic.gmu.edu and michalski@aic.gmu.edu

Abstract

Research on multistrategy task-adaptive learning aims at integrating all basic inferential learning strategies—learning by deduction, induction and analogy. The implementation of such a learning system requires a knowledge representation that facilitates performing a multitype inference in a seamlessly integrated fashion. This paper presents an approach to implementing such multitype inference based on a novel knowledge representation, called *Dynamic Interlaced Hierarchies* (DIH). DIH integrates ideas from our research on cognitive modeling of human plausible reasoning, the Inferential Theory of Learning, and knowledge visualization. In DIH, knowledge is partitioned into a "static" part that represents relatively stable knowledge, and a "dynamic" part that represents knowledge that changes relatively frequently. The static part is organized into type, part, or *precedence* hierarchies, while the dynamic part consists of *traces* that link nodes of different hierarchies. By modifying traces in different ways, the system can perform different *knowledge transmutations* (patterns of inference), such as generalization, abstraction, similization, and their opposites, specialization, concretion and dissimilization, respectively.

Key words: multistrategy learning, inferential theory of learning, knowledge transmutation, generalization, abstraction, similization.

1. Introduction

The development of multistrategy learning systems requires a powerful and easily modifiable knowledge representation that facilitates multitype inference. This is particularly true in the case of *multistrategy task-adaptive learning* (MTL) systems that integrate a whole range of inferential strategies, such as empirical induction, abduction, deduction, plausible deduction, abstraction, and analogy (Michalski, 1990, 1991; Tecuci and Michalski, 1991; Tecuci, 1993). A MTL system adapts a strategy or a combination of strategies to the *learning task*, defined by the available input knowledge, the learner's background knowledge and the learning goal. A theoretical framework for the development of MTL systems has been presented in (Michalski, 1993).

This paper presents basic ideas underlying a knowledge representation proposed for the implementation of a MTL system and its use for implementing multitype inference. This representation, called *Dynamic Interlaced Hierarchies* (DIH), integrates ideas from our research on modeling human plausible inference, the Inferential Theory of Learning and the visualization of knowledge. DIH

encompasses many different forms of knowledge – facts, rules, dependencies, etc., and facilitates knowledge transmutations, described in the Inferential Theory of Learning (ITL) (Michalski, 1993). This paper shows how DIH supports several basic patterns of knowledge change (transmutations), such as generalization, abstraction, similization, and their opposites, specialization, concretion and dissimilization, respectively. These operations are performed on DIH traces, which correspond to well-formed predicate logic expressions associated with a degree of belief.

While our previous work has focused on the visualization of attribute-based representations for empirical induction (Wnek & Michalski, 1991), DIH allows the visualization of structural (attributional and relational) representations. The underlying assumption is that the syntactic structure for representing any knowledge should reflect as closely as possible the semantic relationships among the knowledge components, and facilitate knowledge modifications that correspond to the most frequently performed inferences. An early implementation of this idea was in the ADVISE system, which used three forms of knowledge representation: relational tables, networks and rules (Michalski et al., 1986).

The DIH approach assumes that a large part of human conceptual knowledge is organized into various hierarchies, primarily type, part and precedence hierarchies (see Section 3 for an explanation). Such hierarchies reflect frequently occurring relationships among knowledge components, and make it easy to perform basic forms of inference.

The initial idea for DIH stems from the core theory of human plausible reasoning (Collins & Michalski, 1989; Boehm-Davis, Dontas &

Michalski, 1990). The theory presents a formal representation of various plausible inference patterns observed in human reasoning.

DIH is more fully described in (Hieb & Michalski, 1993).

2. Relevant Research

The core theory of Plausible Reasoning presents a system that formalizes various plausible inference patterns and “merit parameters” that affect the certainty of these inferences. This system combines structural aspects of reasoning (determined by knowledge structures) with parametric aspects that represent quantitative belief and other measures affecting the reasoning process.

Various components of the “Logic of Plausible Reasoning” have been implemented in several systems (Baker, Burstein & Collins, 1987; Dontas & Zemakova, 1988; Kelly, 1988). These implementations used various subsets of the inferences (“statement transforms”) described in the core theory to investigate the parametric aspects of the theory. The implementations demonstrated how the core theory of plausible reasoning can be applied to various domains. DIH specifies a broader set of knowledge transmutations in a general and well-defined knowledge representation. These transmutations are part of a framework for both reasoning and learning.

The organization of concepts into various hierarchies has been proposed as a plausible structure for human semantic memory quite early (Collins & Quillian, 1972). The WordNet project at Princeton University, directed by George Miller, concerns the implementation of an electronic thesaurus using such a memory structure (Beckwith et

al., 1991). WordNet is a very large lexical database with approximately 50,000 different word forms. WordNet divides the lexicon into various categories including nouns, verbs, and modifiers (adjectives and adverbs). Significantly, the nouns are stored in topical hierarchies (both type and part hierarchies), lending support to the DIH representation. However, while WordNet can be used as a source of DIH hierarchies, it does not provide any inferential facilities.

Other relevant research includes the development of the Common Knowledge Representation Language (CKRL), done as part of an ESPRIT project (Morik, Causse & Boswell, 1991). CKRL offers a language in which knowledge can be exchanged between machine learning tools and it uses the set of *most* common representation structures and operators. While CKRL's representation for multistrategy learning seeks to integrate the various representations employed by several different learning programs for communication of knowledge between the machine learning tools, our aim is to develop a representation that facilitates an integration of learning and inference processes.

Semantic network knowledge representation systems, such as the KL-ONE family (Brachman et al., 1991), utilize a large network of relationships between concepts, intermixing different relationships. The hierarchies they use are tangled, in which a concept can have more than one parent. As a consequence, implementing knowledge transmutations, e.g., generalization, is not as easy as in DIH. DIH facilitates such transmutations because it uses only single-parent hierarchies, representing a structuring of a set of entities from a certain viewpoint. In DIH, a concept can belong to different hierarchies, reflecting the fact that a given concept (or object) can

usually be classified from several different viewpoints.

The design of semantic networks is primarily oriented toward facilitating deductive inference, and is not usually concerned with knowledge visualization. The design of DIH is oriented toward facilitating multitype inference and providing a basis for the visual presentation of knowledge. DIH also utilizes a hierarchy of merit parameters to represent probabilistic factors associated with plausible reasoning.

3. Basic Components of DIH

The theory of plausible reasoning postulates that there are recurring patterns of human plausible inference. To adequately represent these patterns, one needs a proper knowledge representation. The DIH approach partitions knowledge into a "static" part and "dynamic" part. The static part represents knowledge that is relatively stable (such as established hierarchies of concepts), and a "dynamic" part that represents knowledge that changes relatively frequently (such as statements representing new observations or results of reasoning). The static part is organized into *type* hierarchies (TH), *part* hierarchies (PH) and *precedence* hierarchies. Precedence hierarchies include several subclasses, specifically, *measure* hierarchies (MH), *quantification* hierarchies (QH) and *schema* hierarchies (SH). The dynamic part consists of *traces* that represent knowledge involving concepts from different hierarchies. Each trace links nodes of two or more hierarchies and is assigned a degree of belief.

These hierarchies are composed of nodes representing abstract or physical entities, and links representing certain basic relationships among the entities, such as "type-of", "part-of" or "precedes". In the "pure" form, these

hierarchies are single parent, that is, no node can have more than one parent. The root node is assigned the name of the class of entities that are organized into the hierarchy from a given viewpoint.

A type (or generalization) hierarchy organizes concepts in a given class according to the "type-of" relation (also called a "generalization" or "kind-of" relation). For example, different types of "animals" can be organized into a "type" hierarchy.

A part hierarchy organizes entities according to a "part-of" relationship. For example, the world, viewed as a system of continents, geographical regions, countries, etc., can be organized into a part hierarchy. While properties of a parent node in the type hierarchy are inherited by children nodes, this does not necessarily hold for a part hierarchy. There are several different part relationships, which include part-component, part-member, part-location and part-substance (Winston, Chaffin and Herrmann, 1987).

To represent relationships among elements of ordered or partially ordered sets, a class of *precedence* hierarchies is introduced. Hierarchies in this class represent hierarchical structures of concepts ordered according to some precedence relation, such as "A precedes B", "A is greater than B", "A has higher rank than B", etc.

There are several subclasses of precedence hierarchies. One subclass is a *measure*

hierarchy, in which leafs stand for values of some physical measurement, for example, weight, length, width, etc., and the parent nodes are symbolic labels characterizing ranges of these values, such as "low", "medium", "high", etc. Figure 1 shows a measure hierarchy of possible values of people's height. Dotted lines indicate a continuity of values between nodes. Arrows indicate the precedence order of the nodes. Another subclass hierarchy is a *belief* hierarchy, in which nodes represent degrees of an agent's beliefs in some knowledge represented by a trace.

Other subclasses of precedence hierarchies include a *rank* hierarchy and a *quantification* hierarchy. A rank hierarchy consists of values representing the "rank" of an entity in some structure, e.g., an administrative hierarchy or military hierarchy. A quantification hierarchy consists of nodes that represent different quantifiers for a set (An example is shown in Figure 2). A quantification hierarchy that is frequently used in commonsense reasoning includes such nodes as "one", "some" (corresponding to the existential quantifier), "most", and "all" (corresponding to the universal quantifier).

Each hierarchy has a heading that specifies its kind (TH, PH, MH, QH or SH) and the underlying concept (or viewpoint) used for the creation of the hierarchy. In addition, the type and part hierarchies also have a *top* node that in the type hierarchies stands for the class of

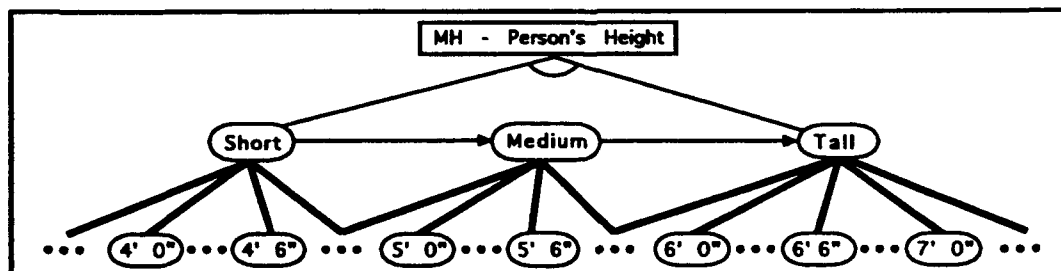


Figure 1: A measure hierarchy of values characterizing people's height.

all entities in the hierarchical structure, and in the part hierarchies for the complete object.

Schema hierarchies (or schema) are structures that indicate which hierarchies are connected in order to express multi-argument concepts or relationships. For example, the schema hierarchy for the concept of "physical-object" can be <shape, size>. This states that an attribute "shape" applies to any object that is a "physical-object" (a node in the "physical-object" hierarchy), and produces a shape value, which is a node in the "shape" hierarchy. The schema hierarchy for the concept of "giving" may be <giver, receiver, object, time> that states that this concept involves an agent that gives, an agent that receives, an object that is being given, and the time when the "giving" occurs. The agents, object and time are elements of their respective hierarchies.

DIH also makes a distinction between structural and parametric knowledge. The structural knowledge is represented by hierarchies and traces that link nodes of different hierarchies. Parametric knowledge consists of numeric quantities characterizing structural elements of knowledge. In DIH, this knowledge is represented via precedence hierarchies of *merit parameters*. The basic merit parameter is a belief measure that characterizes the "truth" relationship of a given component of knowledge representation (a trace), as estimated by the reasoning agent. Other merit parameters include the *forward* and *backward* strength of a dependency, *frequency*, *dominance*, etc. (Collins and Michalski, 1989; Michalski, 1993). In this paper, we will consider only one merit parameter, namely, the belief measure.

The theory of human plausible reasoning (Collins and Michalski, 1989) postulates that

people rely primarily on the structural knowledge, and resort to parametric knowledge when the "structural" reasoning does not produce a unique result. They resist performing uncertain inferences based on only parametric knowledge, and they are not good at assigning a degree of certainty to a statement based only on the combination of the certainties of its constituents, without taking into consideration the meaning of the whole sentence. A reason for this may be that there does not exist a normative model for reasoning under uncertainty that is independent of the structural aspects of knowledge, i.e., its meaning. Plausible reasoning about a problem or question typically involves both structural and parametric knowledge components.

Nodes of a hierarchy are elementary units of the DIH representation. Each node represents some real or abstract entity—a concept, an object, a process, etc. A given entity can be a node in multiple hierarchies, where each hierarchy structures a set of entities from a different viewpoint. The relevant viewpoint is determined by the context of the discourse.

As mentioned earlier, the basic structures in the DIH representation are hierarchies, nodes, traces and schema. Our research on DIH demonstrates that these structures provide a very natural environment for performing basic types of inference on statements. The subsequent sections show how these inferences are performed using the DIH representation.

4. DIH Traces

To describe the DIH knowledge representation, let us start by representing the following statement: "It is certain that some power plants in New York have mechanical

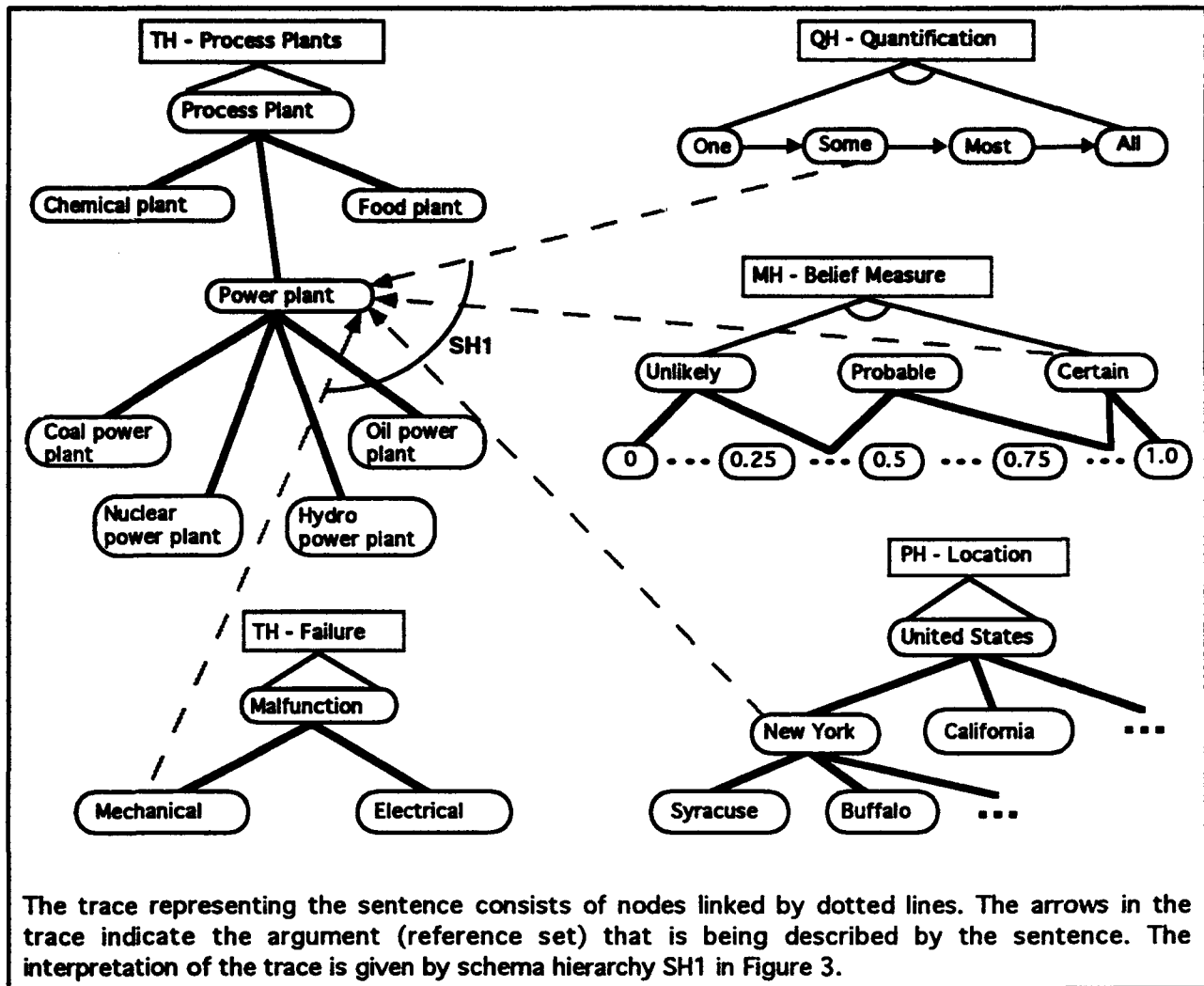


Figure 2: A DIH trace representing the sentence "It is certain that some power plants in New York have mechanical failures."

failures." Figure 2 presents this statement as a trace connecting nodes of five hierarchies: "Process plants" and "Failure", both type hierarchies; "Quantification", the quantification hierarchy; "Location", a part hierarchy; and "Belief measure" a measure hierarchy.

The interpretation of the trace is done on the

basis of the schema hierarchy shown in Figure 3. The schema defines the universe of sentences that can be generated using concepts of these hierarchies, ordered according to the schema.

The convention for the direction of arrows in a trace is that they point from the nodes

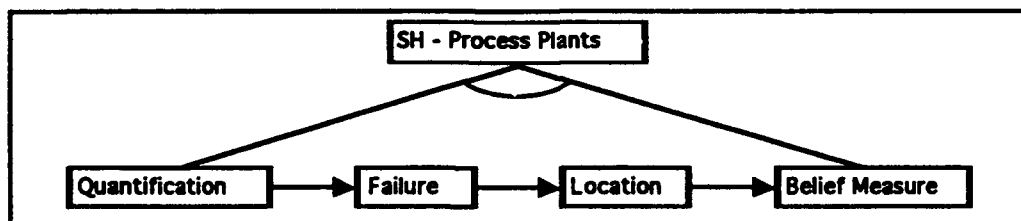


Figure 3: Schema hierarchy SH1.

denoting descriptive concepts to the *argument node* that stands for the set (or individual) being described, called a *reference set*. In this example, the set being described is "Power plant" in the hierarchy of Process Plants, thus the node representing it is the argument node. Other nodes linked by the trace represent descriptive concepts for the argument node. The belief measure takes values from a belief hierarchy, and refers to the entire trace rather than a single node, which is indicated by the schema.

Using the formalism of the annotated predicate logic (Michalski, 1983), this trace can be interpreted as: "(Some)x, [type(x) = Power plant] & [location(x) = New York] & [failure(x) = mechanical]: Belief = 1.0." This statement is a quantified conjunction of several *elementary* statements. An elementary statement expresses one property of the reference node (set), for example, "Location(Power plant) = New York."

In a formal expression of an elementary statement, the reference set ("Power plant") is called an *argument*, the predicate ("Location") is called a *descriptor*, and the value of the descriptor ("New York") is called the *referent*. Thus, an elementary statement is formally expressed in the form "descriptor(argument) = referent".

In Figure 2, the square boxes contain the heading of the hierarchy. The concept specified in the heading is the general descriptor for the hierarchy. The nodes in the hierarchy are possible values of this descriptor.

The schema hierarchy, SH1, in Figure 3 is used for the interpretation of the trace represented in Figure 2. The heading indicates the type of hierarchy (SH: Schema Hierarchy)

and the reference set of the trace. Since the schema hierarchy is a precedence hierarchy, a valid interpretation of the schema requires each of the descriptors in order. Thus the first element of the trace must be from the quantification hierarchy, the second from the failure hierarchy, the third from the location hierarchy and the last from the hierarchy of belief measures. This schema hierarchy is also utilized for examples in Section 4.

Adding knowledge to the DIH representation is done by creating hierarchies and specifying traces that express statements involving nodes of different hierarchies. To allow proper interpretation of a trace, the schema is also specified by indicating relevant descriptors and their order.

DIH allows one to represent complex forms of knowledge, involving different kinds of quantifiers, multi-argument predicates, different types of logical operations on them, and to associate degrees of belief with individual statements. A more complete description of the DIH representation system is given in (Hieb & Michalski, 1993).

5. Multitype Inference in DIH

The core theory of plausible reasoning introduced in (Collins & Michalski, 1989) gives four knowledge transmutation operators (also called transforms) – generalization, specialization, similization and dissimilization. The Inferential Theory of Learning (Michalski, 1993) specifies several additional operators, of which abstraction and concretion are incorporated into DIH. (In (Collins and Michalski, 1989), the abstraction and concretion transmutations were called referent generalization and referent specialization, respectively.)

Transmutation	Symbol	Relevant Hierarchies	Inference Type
Argument Generalization	AGen	Type, Part	Deductive
Argument Specialization	ASpec	Type, Part	Inductive
Quantification Generalization	QGen	Quantification	Inductive
Quantification Specialization	QSpec	Quantification	Deductive
Abstraction	Abs	Type, Part, Precedence	Deductive
Concretion	Con	Type, Part, Precedence	Inductive
Argument Similization	ASim	Type, Part	Analogical
Argument Dissimilization	ADis	Type, Part	Analogical
Referent Similization	RSim	Type, Part, Precedence	Analogical
Referent Dissimilization	RDis	Type, Part, Precedence	Analogical

Table 1: Basic knowledge generation transmutations.

Generalization (specialization) transmutations extend (contract) the reference set. They are done either by argument generalization (specialization) or by quantification generalization (specialization). Argument generalization is accomplished by moving above the node representing the reference set in a type hierarchy. Quantification generalization is accomplished by moving up the quantification hierarchy.

Abstraction (concretion) transmutations decrease (increase) the amount of information about the reference set. A way to accomplish such a transmutation is by moving above the node in the type or part hierarchy that corresponds to a value of some descriptor in the sentence represented by the trace.

Similization (dissimilization) transmutation is done by replacing a node corresponding to the reference set (argument) or a descriptor value (referent) by a node at the same level of hierarchy, which corresponds to a similar (dissimilar) concept within the context of the given hierarchy. In the case of dissimilization, the resulting trace is linked with a negation node, because the generated inference is a negation of the original sentence (Michalski, 1993).

These transmutations can be given a simple conceptual interpretation, if one assumes that nodes at each level of hierarchy are ordered by the relation of similarity, that is, nodes that correspond to similar concepts (in the context of the given hierarchy) are located near each other, and nodes that correspond to dissimilar concepts are placed far away from each other. Such an arrangement is natural for precedence hierarchies. In sum, similization and dissimilization transmutations are performed by sideways node movements, while generalization (specialization) and abstraction (concretion) are performed by upward (downward) node movements.

Table 1 lists all the above knowledge transmutations, specifying their abbreviated name, the relevant hierarchies, and the underlying inference type. The relevant hierarchies are the kinds of hierarchies for which the transmutations are valid. The various kinds of part hierarchies are not shown, but are distinguished in DIH. Additional constraints are necessary in some kinds of part hierarchies to maintain the validity of the transmutation.

Figure 4 presents a schematic diagram illustrating how knowledge transmutations

modify a trace. A dotted line represents a link in a trace. An arrow means that the trace is moving to a new node in the indicated direction by performing the indicated transmutation. The quantification transmutations operate over the entire trace, rather than on a single node, as do the transformations involving the merit parameters.

One form of generalization transmutation moves a node in the quantification hierarchy upward, another form moves a node (argument) in the type hierarchy upward. The "+" indicates a strengthening of a merit parameter, or the movement of the link to a node that is "higher" in the particular merit parameter measure hierarchy. The "-" indicates a weakening of the merit parameter, or the movement of the link down in the hierarchy.

Moving a node in a trace in a manner that corresponds to a deductive inference (Table 1) produces a new trace (statement) with the

same truth status as the original trace. In the case of node movement that corresponds to inductive or analogical inference, the smaller the node movement ("perturbation"), the more plausible the resulting inference.

The Argument Generalization transmutation represents a deductive inference. The abstraction operation is also deductive. In contrast, Argument Specialization, Quantification generalization and Concretion are inductive, because they produce traces (statements) that logically entail the original traces (statements).

The above transmutations can be usually done in a number of different ways, by moving to different alternative nodes. The plausibility of the generated statements depends on additional merit parameters, such as dominance, typicality, multiplicity, similarity, frequency, etc. (Collins and Michalski, 1989). These issues will be the subject of future research.

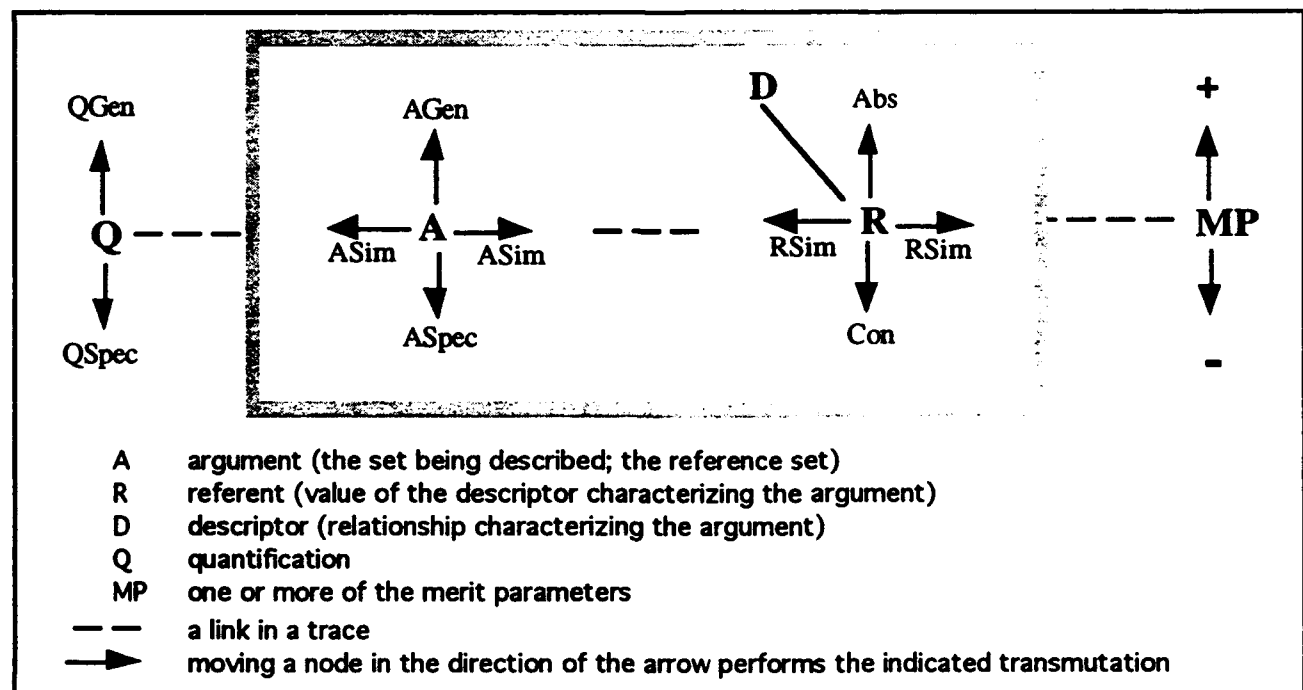
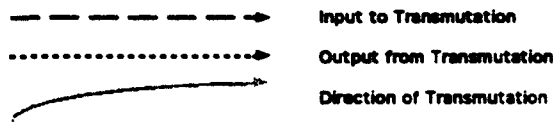


Figure 4: Diagram of knowledge transmutations in DIH.

6. Visualizing DIH-Based Inference

This section illustrates several basic transmutations through a series of self-explanatory examples. These examples involve the same original statement, represented as a trace in Figure 2. Given the original statement, these transmutations generate new statements illustrated by DIH traces in Figures 5 through 12.



The legend above is used for interpreting the following figures. The input statement is the same as that of Figure 2, without the belief measure hierarchy. All of the examples are interpreted according to the schema SH1 shown

in Figure 3.

There are two referents in the input statement. The resulting statements (output) show the results of the given transmutation assuming that there are no merit parameters that assist in the specialization or concretion and that the similization operator finds a single "most similar" node using the descriptors given. The Background Knowledge (BK) is the learner's prior knowledge that is relevant to the learning process.

7. MTL-DIH System

The research on DIH aims at developing a representation that will facilitate all basic inferential strategies and knowledge transmutations to be implemented in the multistrategy task-adaptive learning system (MTL-DIH).

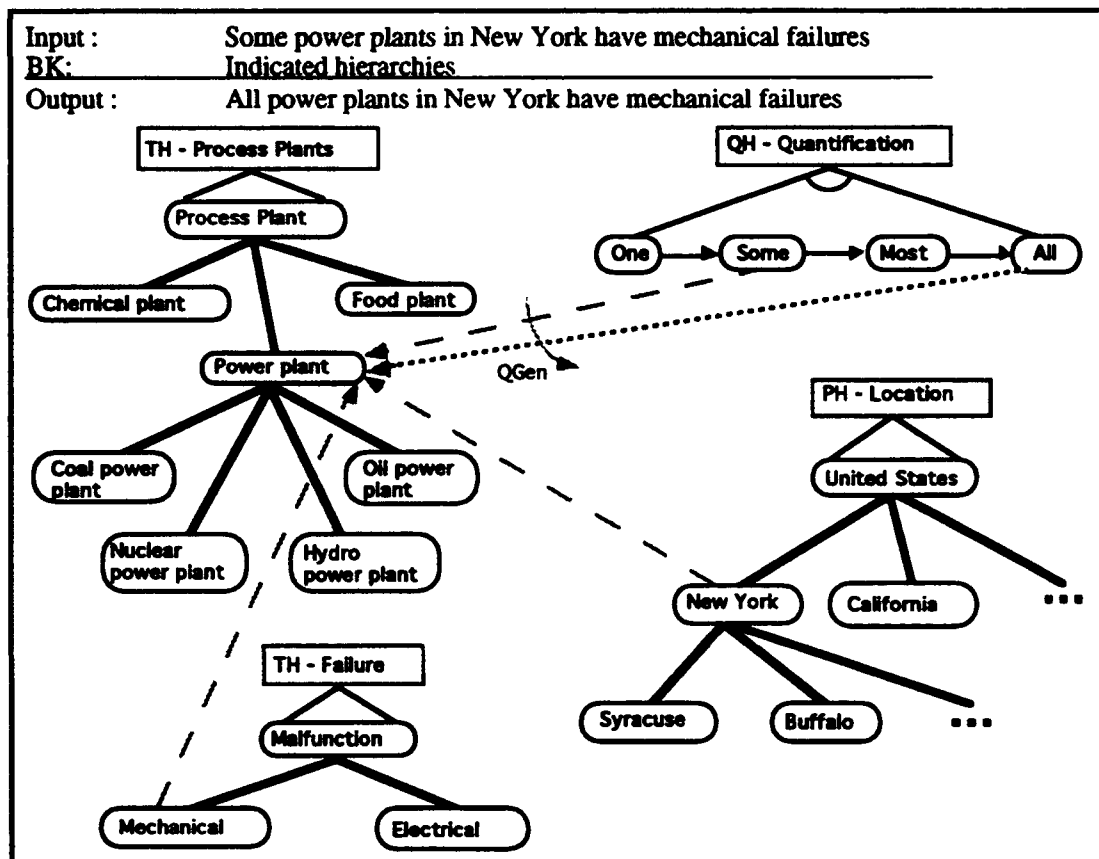


Figure 5: Inductive generalization based on quantification.

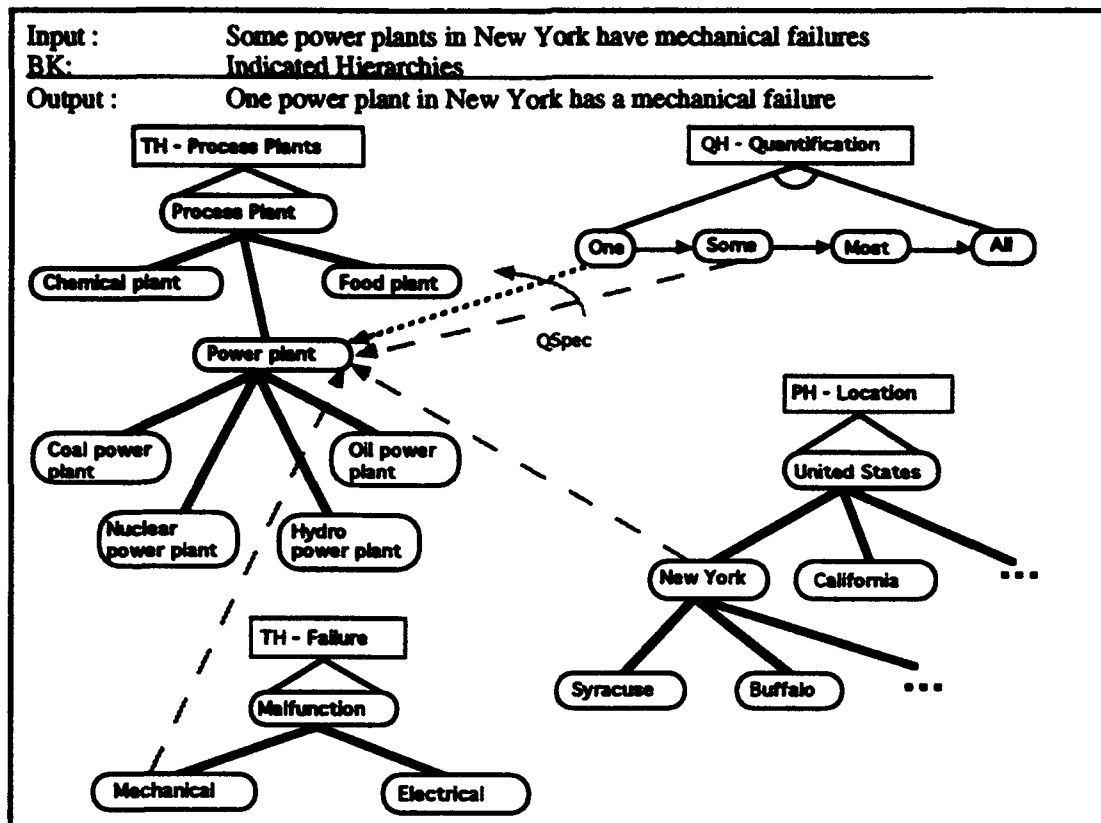


Figure 6: Deductive specialization based on quantification.

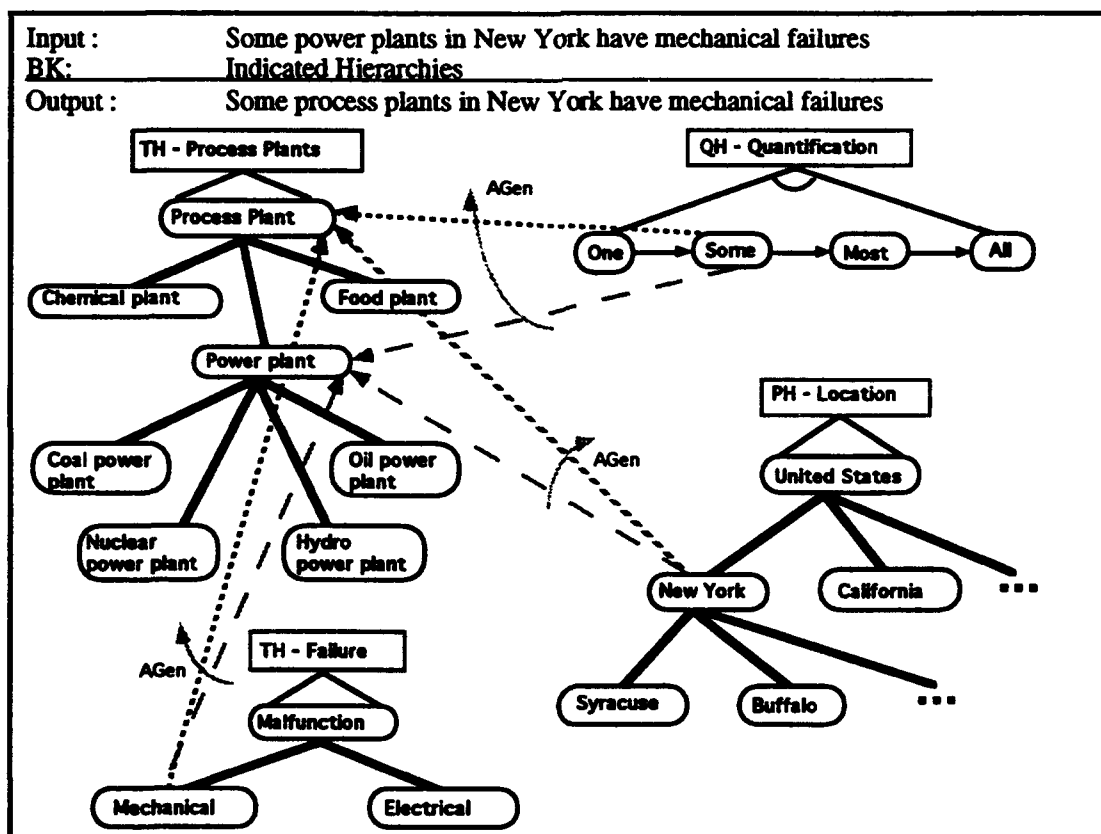


Figure 7: Deductive generalization based on the argument.

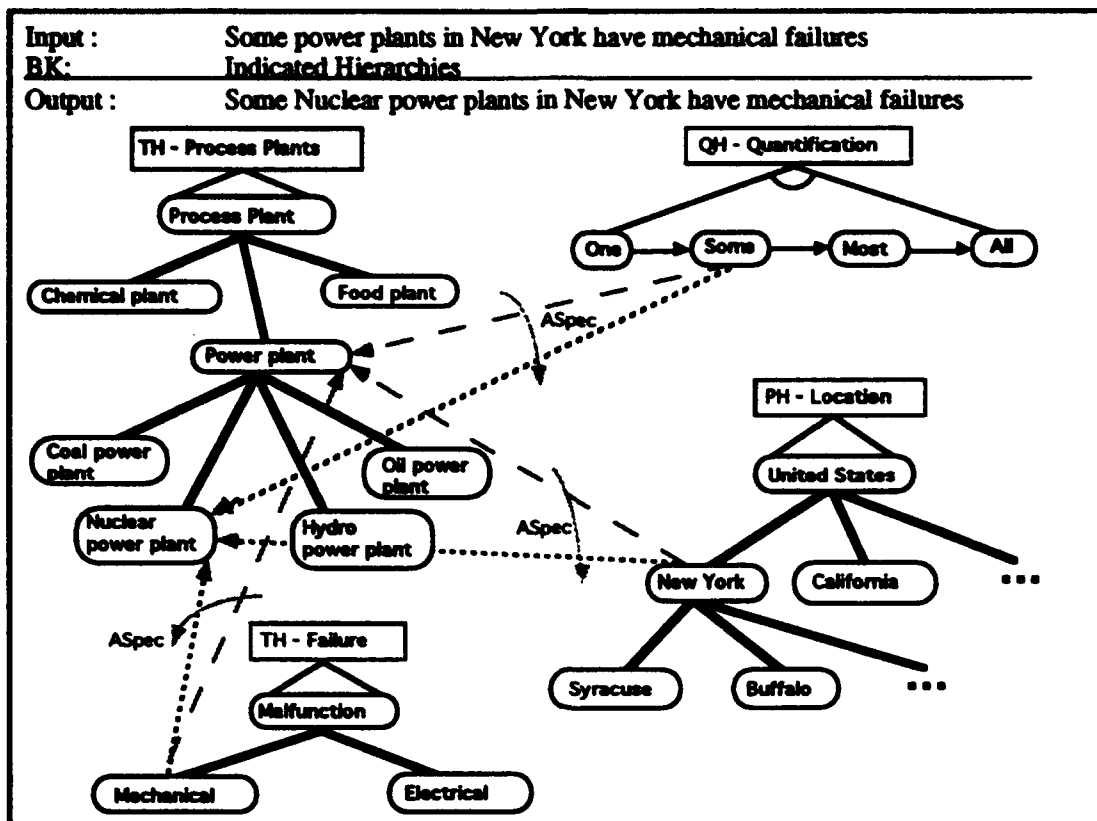


Figure 8: Inductive specialization based on the argument.

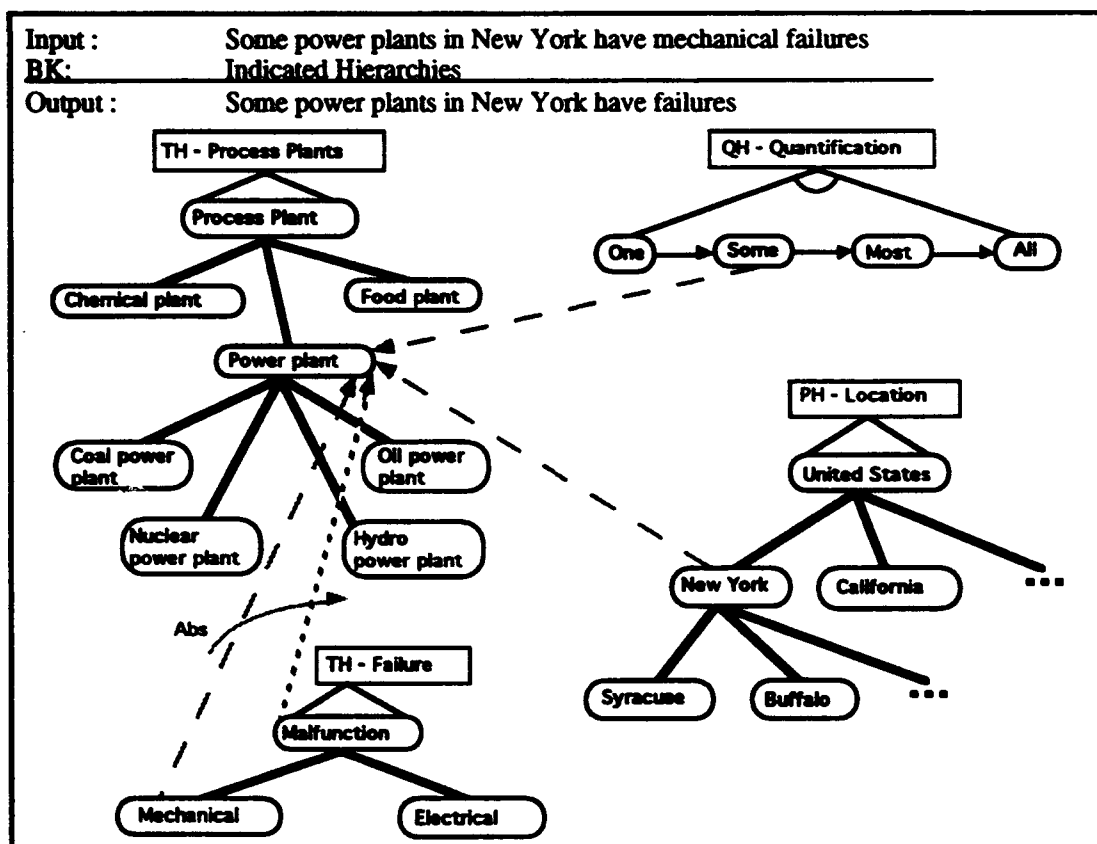


Figure 9: Abstraction transmutation.

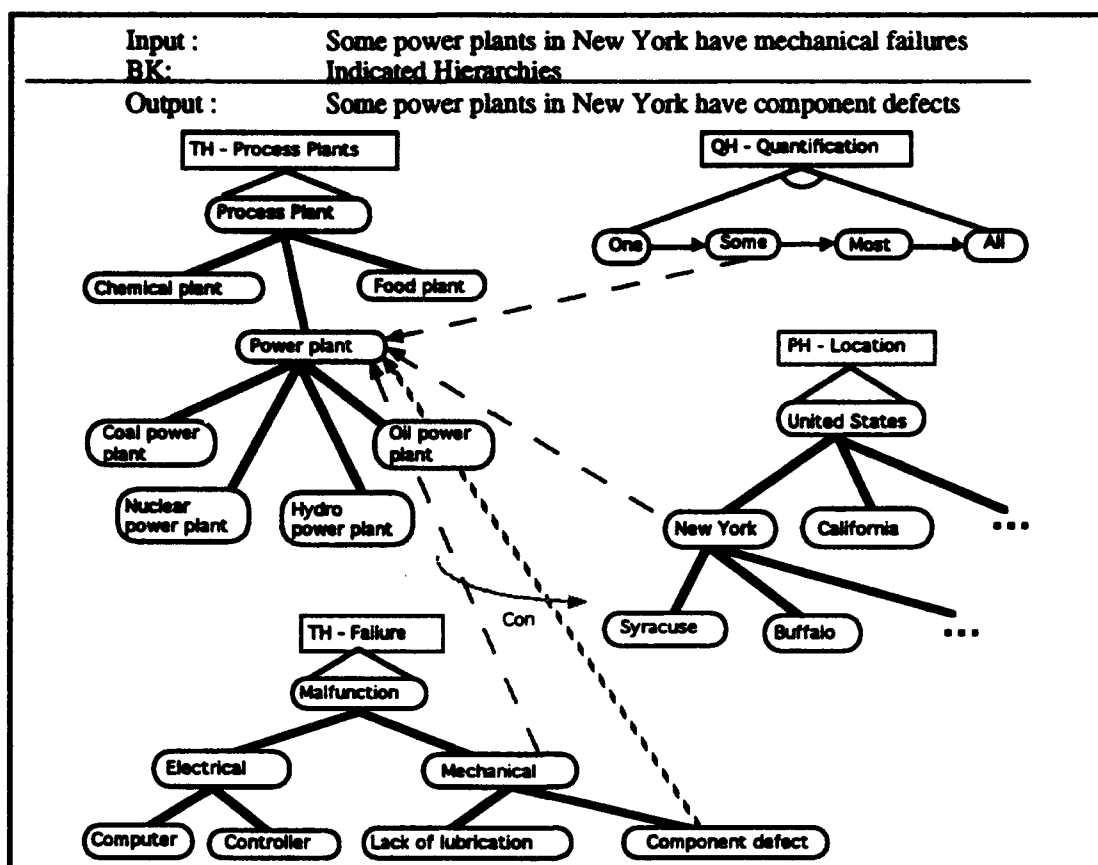


Figure 10: Concretion transmutation.

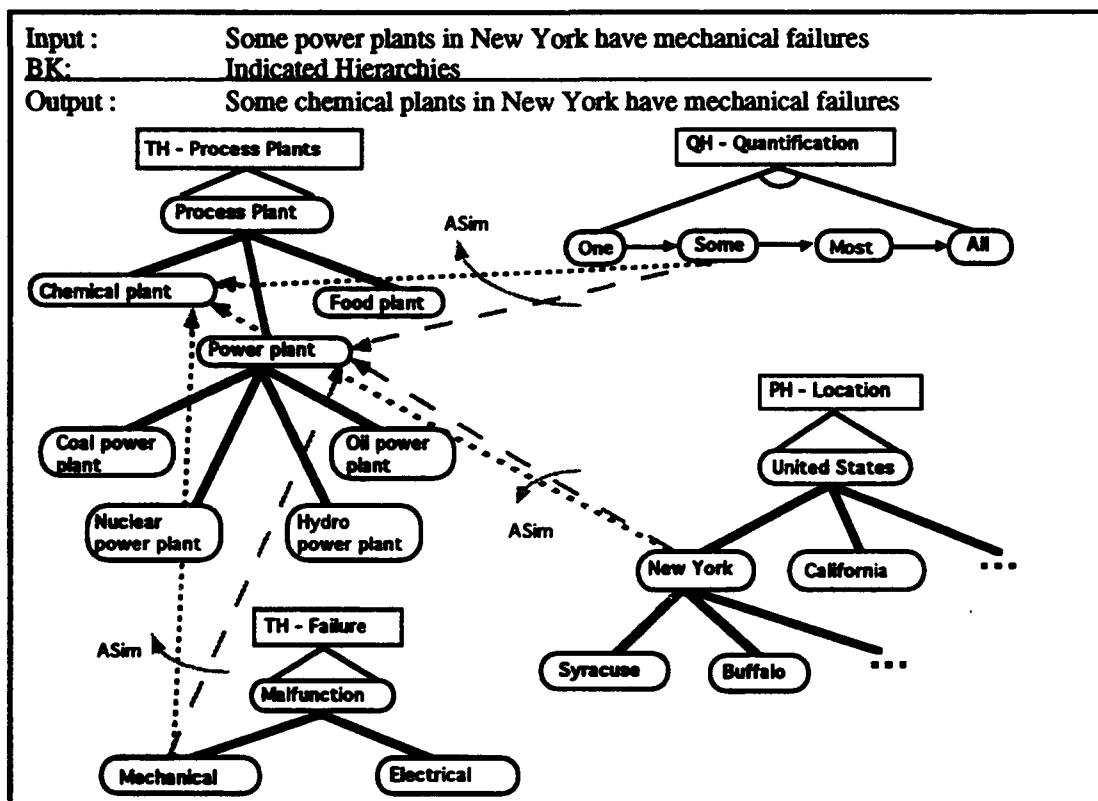


Figure 11: Argument similization transmutation.

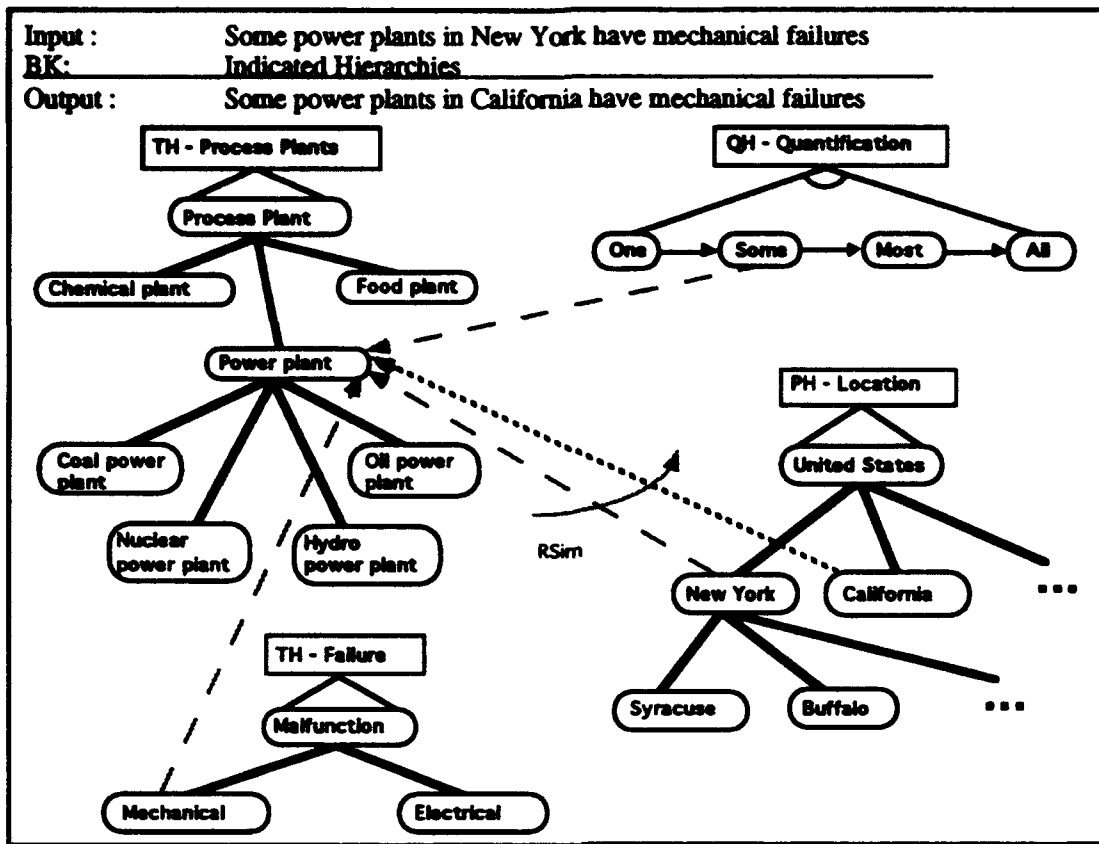


Figure 12: Referent similization transmutation.

Although issues related to the implementation of an MTL system are beyond the scope of this paper, we will briefly outline the basic ideas. We have been pursuing two approaches, MTL-JT, which builds a plausible justification tree to "understand" a user's input (Tecuci, 1993), and a second one, MTL-DIH, based on DIH.

In the MTL-DIH approach, a learning strategy is determined by analyzing the learning task. This analysis relates the input information to the learner's background knowledge and the learning goal. The input information to the system is assumed to be given in the form of logic statements. It can be concept examples, concept descriptions, rule examples, rules or a combination of the above. The system re-represents the input as a trace, or set of traces. Background knowledge is the part of the

learner's prior knowledge that is relevant to the input and the learning goal.

The learning goal specifies criteria characterizing knowledge to be learned. There are different kinds of learning goals, such as to predict new information, to explain the input, to classify a fact or concept instance, to create an abstract description from an operational one or conversely, to create a problem solution or a plan. It is assumed that the learning goal is determined by a teacher or by the control module of the system.

The learning process involves determining the type of relationship between the given input and the background knowledge, and performing a sequence of knowledge transmutations, involving input and background knowledge, to produce knowledge satisfying the learning goal.

8. Summary and Future Research

The DIH knowledge representation presented serves as the basis for implementing multistrategy task-adaptive learning. It builds upon ideas of the Inferential Theory of Learning and the core theory of plausible reasoning. Although it is closely related to the semantic network representation, it represents a significantly different approach, and contains many new ideas that make it particularly useful for representing multitype inference. These include the idea of dividing the knowledge representation into a static part and a dynamic part, the organization of knowledge in which basic forms of inference can be performed via simple trace perturbations, and the introduction of various precedence hierarchies, such as the schema hierarchy, the measure hierarchy, and the quantification hierarchy.

The primary purpose of this paper was to demonstrate how DIH supports several basic knowledge generation transmutations, specifically, generalization, specialization, abstraction, concretion, similization and dissimilization. The first version of DIH has been implemented in Smalltalk, and used as a tool for investigating the interactive display and modification of traces in hierarchies. The visual display of inference is particularly useful in situations that involve traces connecting only a few hierarchies (that is, representing short sentences). To facilitate knowledge visualization, the system has an option to present traces with only a limited number of neighboring nodes, rather than connecting complete hierarchies.

In DIH, the more knowledge structures there are in background knowledge, the easier it is to assimilate new knowledge, or to plausibly

explain input statements. DIH is an efficient, representation, because most knowledge modifications consist of forming or changing traces, without affecting the established hierarchies.

Many issues remain to be addressed in future research. Among these issues are the representation of more complex forms of knowledge—mutual implications, various types of dependencies, temporal and spatial knowledge, and the development of methods for determining the affect of merit parameters on the reasoning process.

Acknowledgments

The authors thank Eric Bloedorn, Mark Burnstein, David Hille and Ken Kaufman for their thoughtful comments on this paper.

This research was conducted in the Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the National Science Foundation under grant No. IRI-9020266, in part by the Advanced Research Projects Agency under the grant No. N00014-91-J-1854, administered by the Office of Naval Research and the grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under grant No. N00014-91-J-1351.

References

- Baker, M., Burstein, M.H., and Collins, A.M., "Implementing a Model of Human Plausible Reasoning," *Proceedings of the Tenth International Joint Conference of Artificial Intelligence*, pp. 185-188, Los Altos, CA: Morgan Kaufman, 1987.
- Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A. and Borgida, A., "Living with CLASSIC: When and How to Use a KL-ONE-Like Language," *Principles of Semantic Networks - Explorations in the Representation of Knowledge*, J.F. Sowa (ed.), Morgan Kaufmann Publishers, 1991.

- Beckwith, R., Fellbaum, C., Gross, D., and Miller, G.A., "WordNet: A Lexical Database Organized on Psycholinguistic Principles," *Using On-line Resources to Build a Lexicon*, U. Zernick (Ed.), Hillsdale, NJ: Erlbaum, 1991.
- Boehm-Davis, D., Dontas, K., and Michalski, R.S., "A Validation and Exploration of the Collins-Michalski Theory of Plausible Reasoning," *Reports of the Machine Learning and Inference Laboratory*, MLI 90-5, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1990.
- Collins, A., and Michalski, R.S., "The Logic of Plausible Reasoning: A Core Theory," *Cognitive Science*, Vol. 13, pp. 1-49, 1989.
- Collins, A., and Quillian, M.R., "How To Make A Language User," *Organization of Memory*, E. Tulving and W. Donaldson (eds.), New York: Academic, 1972.
- Dontas, K., and Zemakova, M., "APPLAUSE: An Implementation of the Collins-Michalski Theory of Plausible Reasoning," *Proceedings of the Third International Symposium on Methodologies for Intelligent Systems*, Torino, Italy, 1988.
- Hieb, M.R. and Michalski, R.S., "A Knowledge Representation System Based on Dynamic Interlaced Hierarchies: Basic Ideas and Examples," *Reports of the Machine Learning and Inference Laboratory*, MLI 93-5, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1993.
- Kelly, J., *PRS: A System for Plausible Reasoning*, M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, 1988.
- Michalski, R. S., "A Theory and Methodology of Inductive Learning," Chapter in the book, *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. Carbonell and T. Mitchell (Eds.), TIOGA Publishing Co., Palo Alto, 1983, pp. 83-134.
- Michalski, R.S., "Towards a Unified Theory of Learning: Multistrategy Task-adaptive Learning," *Reports of the Machine Learning and Inference Laboratory*, MLI 90-1, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1990.
- Michalski, R.S., "Inferential Learning Theory as a Basis for Multistrategy Task-Adaptive Learning," *First International Workshop on Multistrategy Learning*, pp. 3-18, Harpers Ferry, West Virginia, 1991.
- Michalski, R.S., "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning," in *Machine Learning: A Multistrategy Approach, Volume 4*, R.S. Michalski & G. Tecuci (Eds.), Morgan Kaufmann Publishers, 1993.
- Michalski R. S., Baskin A. B., Uhrík C., Channic T., Borodkin S., Boulanger A. G., Rodewald L., and Reinke, "A Technical Description of the ADVISE.1 Meta-Expert System that Integrates Multiple Knowledge Representations and Learning Capabilities," ISG 86-8, Department of Computer Science, University of Illinois, Urbana, 1986.
- Morik, K., Causse, K., and Boswell, R., "A Common Knowledge Representation Integrating Learning Tools," *First International Workshop on Multistrategy Learning*, pp. 81-96, Harpers Ferry, West Virginia, 1991.
- Tecuci, G. & Michalski, R.S., "A Method for Multistrategy Task-adaptive Learning Based on Plausible Justifications," in Birnbaum, L., & Collins, G. (Eds.) *Machine Learning: Proceedings of the Eighth International Workshop*, San Mateo, CA, Morgan Kaufmann, 1991.
- Tecuci, G., "An Inference-Based Framework for Multistrategy Learning" in *Machine Learning: A Multistrategy Approach, Volume 4*, R.S. Michalski & G. Tecuci (Eds.), Morgan Kaufmann Publishers, 1993.
- Winston, M.E., Chaffin, R., and Herrmann, D., "A Taxonomy of Part-Whole Relations," *Cognitive Science*, Vol. 11, pp. 417-444, 1987.
- Wnek, J. and Michalski, R.S., "An Experimental Comparison of Symbolic and Subsymbolic Learning Paradigms: Phase I - Learning Logic-style Concepts," *First International Workshop on Multistrategy Learning*, pp. 324-339, Harpers Ferry, West Virginia, 1991.

Better Learners Use Analogical Problem Solving Sparingly

Kurt VanLehn
Learning Research and Development Center
University of Pittsburgh
Pittsburgh, PA 15260
VanLehn@cs.pitt.edu
(412)624-7458

Randolph M. Jones
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Av.
Ann Arbor, MI 48109-2110
rjones@engin.umich.edu
(313)764-0683

Abstract

When solving homework exercises, human students often notice that the problem they are about to solve is similar to an example. They then deliberate over whether to refer to the example or to solve the problem without looking at the example. We present protocol analyses showing that effective human learners prefer not to use analogical problem solving for achieving the base-level goals of the problem, although they do use it occasionally for achieving meta-level goals, such as checking solutions or resolving certain kinds of impasses. On the other hand, ineffective learners use analogical problem solving in place of ordinary problem solving, and this prevents them from discovering gaps in their domain theory. An analysis of the task domain (college physics) reveals a testable heuristic for when to use analogy and when to avoid it. The heuristic may be of use in guiding multistrategy learners.

Keywords: incomplete theories, human skill acquisition, multistrategy learning, protocol analysis.

1 When To Use Analogical Problem Solving?

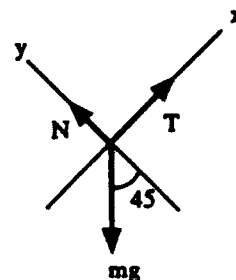
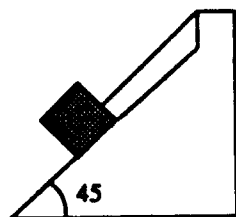
When doing homework exercises, human learners often notice that the problem they are about

to solve is similar to an example, then deliberate over whether to refer to the example or to solve the problem without its aid. As one of our subjects said, "this looks very much like the one I had in the examples. Okay. Should I just go right to the problem, which I distinctly remember? Or should I try to do it without looking at the example?" A multistrategy machine learning program could face the same decision. The objective of this paper is to find out what heuristics good human learners use for deciding whether to do analogical problem solving, then determine when those heuristics would be good for a machine learning program to use.

Because we use protocol data, the only evidence we have of analogical problem solving is episodes where a person explicitly refers to an example, typically by flipping pages in a textbook in order to expose the page on which the example is printed. Thus, analogical problem solving, in this paper, means the process of referring to a written example rather than a mentally held one. As will be seen later, nothing in our conclusions relies on this restriction, so the results may apply to analogies that refer to mental examples (or cases?) as well as written ones.

The protocol data come from subjects learning Newtonian physics. The subjects worked with textbook physics problems and examples, such as the one in figure 1. The protocol data were collected as part of a study by Chi, Bassok, Lewis, Reimann & Glaser (1989). The subjects

Problem: The figure on the left below shows a block of mass m kept at rest on a smooth plane, inclined at an angle of 45 degrees with the horizontal, by means of a string attached to the vertical wall. What are the magnitudes of the forces acting on the block?



Solution:

- (1) We choose the block as the body.
- (2) The forces acting on the block are shown in the free-body diagram on the right.
- (3) Since the block is unaccelerated, we obtain $T + N + mg = 0$.
- (4) It is convenient to choose the x-axis of our reference frame to be along the incline and the y-axis to be normal to the incline.
- (5) With this choice of coordinates, only one force, mg , must be resolved into components.
- (6) The two scalar equations obtained by resolving mg along the x- and y-axes are:

$$T - mg \sin 45 = 0 \quad \text{and} \quad N - mg \cos 45 = 0.$$
- (7) From these equations, T and N can be obtained if m is given.

Figure 1: A physics example, with line numbers added

were 9 college students selected to have similar backgrounds (Chi & VanLehn, 1991). The subjects first refreshed their mathematical knowledge by studying the first 4 chapters of Halliday & Resnick (1981), a popular physics textbook. They then studied the expository part of chapter 5, which introduces the basic principles of Newtonian mechanics, its history and some classic experiments. Student were tested at this point and had to re-study parts of the material that they did not understand. After they had mastered the mathematical prerequisites and the basic principles, they studied 3 examples and solved 19 problems while talking aloud. They were allowed to refer to the examples at any time while solving problems, but they were not allowed to refer to their own previous problem solutions. The 9 subjects' protocols, which averaged 5 hours each, are the raw data for the findings reported here. They contain many instances of analogical problem solving. The goal is to discover which ones helped learning and which ones hurt it.

We used a contrastive protocol analysis tech-

nique pioneered by Chi et al. (1989). The basic idea is to split the subjects into two groups—effective learners and ineffective learners—then determine what the effective learners did differently from the ineffective learners. Because the students were trained to have the same prerequisite knowledge, the scores on their problem solving reflect their learning rate during example studying and problem solving. The 4 highest scoring subjects constitute the effective learners (called Good solvers by Chi et al.), and the 4 lowest scoring subjects constituted the ineffective learners (called Poor solvers). The middle subject's protocol was not analyzed (until later: see below).

The next section presents a learning mechanism and argues that it is the main source of learning by subjects in this study. The argument uses new protocol analyses as well as analyses published earlier. With this as background, the subsequent sections present the main result, which is that effective learners use analogical problem solving sparingly. A discussion section

speculates on why this policy was better for human solvers in this experiment, and suggests conditions under which this policy would be good for any multi-strategy learner.

2 Gap Filling

Given that errors are used to determine when learning was not effective, a direct way to uncover dominant learning mechanisms is to examine the subjects' errors. If errors of a certain type are much less common among Good solvers than Poor solvers, we can assume that a learning mechanism employed by the Good solvers and not the Poor solvers is reducing those errors. From the characteristics of such errors we can infer the characteristics of the learning processes. We classified errors into 5 types, which are listed below:

- *Inappropriate analogies.* Sometimes subjects fetched an example that was inappropriate for the problem being solved. At other times, subjects fetched appropriate examples but applied them in inappropriate ways. Both types of errors are classified as inappropriate analogies.
- *Gap errors.* Subjects often lacked a piece of physics knowledge, such as the fact that the tension in a string is equal to the magnitude of a tension force exerted by that string. Sometimes errors would occur when the subject reached an impasse caused by their lack of knowledge, and used some ineffective repair strategy (VanLehn, 1990) to work around it. At other times, the gap would cause an error (such as a missing minus sign) without the subject ever becoming aware of the gap.
- *Schema selection errors.* All subjects knew several methods or schemas for solving physics problems. One method was to draw forces, generate equations and solve the equations. Another method was simply to generate equations that contained the sought and/or known quantities without considering what forces or other physical

Table 1: Mean errors per subject for each error category

Error type	Good	Poor
Inappropriate analogies	1.00	2.25
Gap errors	**0.25	**7.75
Schema selection	0.50	1.75
Math errors	0.25	0.75
Miscellaneous errors	1.25	1.75
Totals	**3.25	**14.25

quantities might be present. On some problems, subjects chose the equation-chaining schema instead of the force schema, and this caused them to answer the problem incorrectly.

- *Mathematical errors.* A typical mathematical error was to confuse sine and cosine, or to drop a negative sign.
- *Miscellaneous errors.*

The error classification was done separately by two coders, with an intercoder reliability of 82%. Differences were reconciled by collaborative protocol analysis.

Table 1 shows the average number of errors of each type per subject. Although the Good solvers had fewer errors than the Poor solvers in every category, the difference was significant only for gap errors ($t(6) = 5.36, p < .01$). Moreover, the difference was quite large (3.8 standard deviations), and accounts for most (68%) of the difference in the total error rates of the Good and Poor solvers.

These results suggest that Good solvers were more effective learners than Poor solvers because they employed some kind of learning process that filled in the gaps in their knowledge.¹ There are many kinds of mechanisms in the literature that

¹An alternative explanation is that the Good solvers never had the gaps because they learned the knowledge before studying the examples. Analyses of pre-test data (Chi et al., 1989), the instructional material (VanLehn, Jones & Chi, 1991) and the subjects' backgrounds (Chi & VanLehn, 1991) fail to support this interpretation of the data.

can detect and rectify incomplete domain theories. For handy reference, let us refer to the process(es) that Good solvers use as *gap filling* even though we do not know what it is. Table 1 suggests that gap filling is the main learning process that differentiates effective from ineffective learners in this study.

This suggestion is consistent with findings from Chi et al.'s (1989) analysis of the same data. They found that during example studying, Good solvers tended to thoroughly explain the examples to themselves, while the Poor solvers tended to read them rather casually. Further examination of the protocols suggested that self-explanation consisted of actually rederiving the lines of the solution (VanLehn, Jones & Chi, 1991). If the main learning process is gap filling, then this method of studying the example should cause the subjects to detect gaps in their knowledge. If a piece of physics knowledge is required for deriving a line of the example's solution, and students lack that knowledge, then they will be unable to fully explain the line. The resulting impasse might cause them to seek the missing knowledge and fill their gap. Thus, the gap-filling hypothesis is consistent with the finding that Good solvers self-explain examples more than Poor solvers. Moreover, it explains why self-explanation causes better learning (VanLehn & Jones, in press-b).

The computational sufficiency of gap-filling has been tested by implementing simulation of human learning, called Cascade, that is based a particular gap filling mechanism and comparing Cascade's behavior to the protocols (VanLehn, Jones & Chi, 1991; VanLehn & Jones, 1991; in press-a). Gaps can cause Cascade to reach impasses (i.e., be unable to achieve a goal) while trying to solve problems or rederive examples. When Cascade's "official" domain knowledge is insufficient to achieve a goal, it tries to apply overly general knowledge that captures regularities common to many types of scientific and mathematical problem solving. For instance, one overly general rule is that scientific concepts often correspond roughly to common sense concepts. Cascade uses this rule during a problem where a block rests on a spring. Cascade lacks

the knowledge that a compressed spring exerts a force on the objects at its ends, so it reaches an impasse. The overly general rule applies, because Cascade knows that springs push back when you push on them. The overly general rule justifies creating an instance of a scientific concept ("force") because it involves the same objects as an instance of a lay concept ("push back"). As a side-effect of the application of this overly general knowledge, a new domain rule is proposed: If a block rests on a spring, the spring exerts a force on it. If this rule is used successfully enough times, it becomes a full-fledged member of the domain theory. In this fashion, Cascade fills gaps in its domain knowledge.

Cascade's behavior compares well with both aggregate findings (VanLehn, Jones & Chi, 1991) and individual protocols (VanLehn & Jones, 1993). This establishes that with plausible assumptions about subjects' prior knowledge, there is enough information present in the environment to allow a gap filling process to learn everything that the Good solvers learn, to do so without implausibly large computations, and to generate outward behavior that is similar to the subjects' behavior. In short, gap filling is a computationally sufficient account for the Good solvers' learning.

None of the results show that gap filling is the *only* learning process going on. There could be others as well. However, Table 1 suggests that gap filling is the most important learning process, because it accounts for most of the difference in the learning of the Good and Poor solvers.

3 Avoiding Analogical Problem Solving

There is already some evidence that the Good solvers avoid analogical problem solving. This section reviews those findings, then tries to ascertain whether this is a just a correlation or whether avoiding analogy actually causes more effective learning.

Chi et al. (1989) counted episodes of analogical problem solving during the first 3 problems.

They found that Good solvers used analogy only 2.7 times per problem, whereas the Poor solvers used analogy 6.7 items per problem. Thus, the Good solver use analogical problem solving less often than the Poor solvers. Chi et al. also found that the Good solvers used analogy in a more focused way. When the Good solvers referred to an example, they tended to jump into the middle of it and read only a few lines (1.6 lines per episode, on average). The Poor solvers tended to start at the beginning of the example and read the whole thing or until they found something they could use (13.0 lines per episode). This suggests that Good solvers are basically solving the problem on their own, but they occasionally use analogical problem solving to get specific information from the example. The Poor solvers, on the other hand, seem to use analogical problem solving instead of regular problem solving. These findings indicate a correlation between effective learning and avoiding of analogy, but it is not clear which way the causality runs.

Our first hypothesis was that the Poor solvers used more analogical problem solving because they lacked domain knowledge so they had to refer to the example if they were to make any progress. Cascade embedded this hypothesis. It did analogy (called transformational analogy in earlier reports) only when it reached an impasse (VanLehn, Chi & Jones, 1991; VanLehn & Jones, in press-a). On this account, the Chi et al. correlation is due to ineffective learning causing analogy.

However, when we fitted Cascade to individual protocols, we found that we sometimes had to force it to do analogy even though it had the knowledge to do regular problem solving (VanLehn & Jones, 1993). While simulating all 9 subjects, Cascade used analogy 231 times, and 196 of these were caused by impasses while 35 (15%) were caused by our intervention. If we believe the modeling, then these 35 analogies were "optional" in that the subjects did not have to do them. They could have used their knowledge of physics principles instead. In most of these cases (30 of 35), the subjects copied the example's force diagram rather than generate their own. Copying the force diagram was also fre-

quent among the 196 impasse-driven analogies.

Upon reflection, it occurred to us maybe some of these supposedly impasse-driven analogies were not actually caused by trying to generate forces, failing, and reaching an impasse. If this were the case, then one would expect the analogy to yield new knowledge about the missing force (or whatever the missing knowledge was), thus filling the gap and allowing the person to draw their own force diagram the next time it was needed. We examined all 196 cases of analogy and found no cases where this kind of learning occurred. If a person had an gap that caused an impasse-driven analogy, then they would use analogy on every subsequent occasion (if any) when that piece of knowledge was required. It could be that what people learn from such an impasse is that "analogy works here," so they continue to use it. However, it could also be that our modeling was incorrect, and they never had such any impasses for that gap. Instead, when they go to certain sections of the problem (typically, the force diagram), they would use analogy without ever considering using their domain knowledge. Perhaps some of those 196 cases of impasse-driven analogy were really optional analogies. Indeed, two of the subject never tried to draw a force diagram on their own—they always copied an example's diagram.

While investigating the gap-filling hypothesis, we discovered additional support for this conjecture. According to the gap-filling hypothesis, gaps in the textbook become gaps in the student's domain knowledge, which cause errors until they are detected and remedied. In order to check this story, we carefully analyzed the first 5 chapters of Halliday and Resnick (1981) and discovered 9 pieces of knowledge that are required by the problems and are not in the text (VanLehn & Jones, in preparation). Using Cascade, for each of the 9 subjects, we located the places in the protocols where the 9 pieces of knowledge could appear if they were known, or cause errors if they were unknown. For each of the 9 pieces of knowledge, we created a chart, such as the one shown in Table 2, that summarizes what happened at each possible occurrence of the gap. The particular piece of knowledge ref-

erenced by Table 2 is "Projecting a vector onto the negative portion of an axis yields a negative formula." This piece of knowledge is relevant 5 times during example studying and 16 times during problem solving. At each place, for each subject, we classified the protocol fragment into one of the categories shown below (the symbol in parentheses corresponds to the code used in Table 2).

- (E) The subject omitted use of the knowledge, which resulted in an error.
- (O) The subject omitted use of the knowledge, but no error occurred. For instance, one sign error might compensate for another.
- (blank) During example studying, the subject did not explain the part of the example where this piece of knowledge would be used. During problem solving, the subject used analogical problem solving to avoid the line of reasoning that would use the piece of knowledge.
- (U) The subject used the piece of knowledge without hesitation or other signs of unusual processing.
- (L) The subject seemed to learn the knowledge. Episodes received this code if the subjects expressed puzzlement or commented on their lack of knowledge, but eventually came up with the right action (e.g., writing a negative sign). For instance, subject P2 overlooked the first minus sign in the first example, but on the second minus sign she said, "Hmm, why is [it] minus? Uah Huh. . . . Because these axis are starting here so this is minus." She then went back to the first minus sign and said, "How about the X's. It should also be a minus. Yah, that was a minus." Subject S102 paused after seeing the second minus and said, "Negative W. . . . It's because it's going in a negative direction it points. . . they give it a negative value [if] it's below the Y-axis. I mean the X-axis." Subject P1 (quoted at length in

VanLehn, Jones & Chi 1991), took several garden paths before discovering the correct rule for explaining the minus sign, which is clear evidence of her lack of knowledge. However, her verbal behavior at the time of the discovery was just as brief and cryptic as the verbal behavior of P2 and S102. Such limited verbal evidence is typical of discovery events in protocol data (VanLehn, 1991; Siegler & Jenkins, 1989). They nonetheless seem to reliably mark transitions in the subjects' knowledge.

- (R) The subjects' verbal behavior indicates that they are learning the piece of knowledge, but they used it at least one before. We believe these are cases of relearning.
- (?) Protocol missing.

The blanks in the problem solving part of Table 2 shows that for this piece of knowledge, many gaps are not detected because the student used analogical problem solving. When we constructed similar analyses for all 9 gaps and all 9 subjects, we found that of the 81 (= 9x9) cases where a piece of knowledge could be learned, in 44 cases (54%) the subject avoided *all* places where the gap could be detected (as did S109 in Table 2). This analysis clearly indicates that analogical problem solving is thwarting gap filling by avoiding lines of reasoning that would cause the gap to be detected.

This finding makes intuitive sense. Most problems can be solved by a 4 step process: select some objects as the "bodies" (line 1 of Figure 1), draw a diagram for each body showing the forces acting on it (lines 2 and 3 of Figure 1), produce a set of equations (line 6 of Figure 1), then solve the equations for the sought quantity (omitted in Figure 1). The last step cannot usually be replaced by analogy because the problems seldom seek the same quantities. However, the first 3 steps can often be achieved by analogy. The student can find an analogous problem and copy either its force diagram, its equations or both. A student who does this avoids using the force laws (which generate forces) and Newton's laws (which generate the equations). Missing physics

Table 2: Places where the negative-projection rule could be used

Subj.	Examples					Problems															
	1	2	3	4	5	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P2		L			U	?	?	U	U	U	U	U	U	U	U	U	U	U	U	U	U
S101								L	L			O	U			U					U
P1	L	L	U	O	U	U		U	U		U	U		U	U		U		U	U	
S110								O					U	O	O	U	O			U	R
S102		L				U							U	U	O		U				
S109																					
S105	U							U			U		U		U		U				E
S103									E					O	E						
S107	U							E	R			U		U	U	O	U				

knowledge can remain undetected as long as one uses analogy to copy force diagrams and equations. To put it bluntly, analogical problem solving often preserves ignorance.

We can now understand part of Chi et al.'s finding about the use of analogy by Good and Poor solvers. The Poor solvers displayed more episodes of analogical problem solving and read more lines during each episode because they generally avoided generating their own forces and equations by copying them from the examples. This is not just a coincidence, but seems to have caused them to learn much less than they would otherwise. On the other hand, the Good solvers generally tried to generate their own forces and equations and only referred sporadically and briefly to the examples. Thus, they could still detect their gaps and remedy them.

4 Using Analogy Sparingly

We suspect that the Good solvers' use of analogy does more than just allow gap-filling to operate. It may actively aid gap-filling by helping to both detect and fill gaps. This section presents a few pieces of protocol data to support our conjecture. However, more data are clearly required.

There were 6 cases in the protocols where, according to the analysis above, knowledge was learned during problem solving. During 4 of the 6 episodes, the subject clearly referred to an example. Although there are only 4 cases, they are different enough that they begin to show that

analogy can assist gap filling in several ways. To illustrate this, each of the 4 cases will be presented.

In one case, subject P2 reached an impasse and filled it with the aid of analogy. The subject was given a problem where water in a tube supported a block and kept it from falling. The subject quickly recognized that this problem is similar an example where a block hung from a string which prevented it from falling. However, she did not refer to the example but instead went to work on the problem by drawing its forces. Eventually she realized that she needed to know whether the water exerted a force on the block. (The textbook had never mentioned pressure forces, and the subject never used the word "pressure," so apparently she lacked knowledge of this kind of force.) She thought that there might be a force analogous to the tension force in the string, but she wasn't sure. At this point she referred to the example, presumably in order to ascertain its similarity to the problem. She eventually decided that it was okay to assume a force analogous to the tension force. This seems to be a case of an analogy assisting in the formulation of a new physics conjecture that both filled a gap in the subject's knowledge and resolved a problem solving impasse. ACT* (Anderson, 1990) and other theories claim that analogy is often used to resolve impasses and thereby acquire new knowledge.

In another case, subject S105 generated forces for a problem without referring to any exam-

ples. However, he failed to draw one force (the surface normal—a notoriously unintuitive force). He had never produced that force in earlier problem solving, nor had he self-explained the line in the examples that was intended to teach it. Thus, we assume he had a knowledge gap. After drawing forces, the subject fetched an example, viewed its force diagram, said, “That’s the force I wasn’t thinking of,” and drew a normal force on his diagram. Thereafter, the subject regularly drew normal forces on his diagrams. In this case, analogy was used to *check* a step in the problem solving, and that revealed a knowledge gap. Whereas the preceding case illustrates how analogy can help in filling gaps, this case illustrates how analogy can help in detecting gaps. There was a second case of analogy-based checking causing detection of a gap, but it will not be presented here. These cases are consistent with a finding of Chi et al. (1989), who classified analogical episodes as either reading, checking or copying. Good solvers had many fewer episodes of reading and copying than Poor solvers, but they actually had more episodes of checking.

In the last case of learning while doing analogical problem solving, the subject was engaged in a mixture of analogical problem solving and self-explanation. Subject S101 apparently did not know that projection onto the negative part of an axis yields a negative formula. He said, “I’m trying to figure out why these are negative,” referring to two negative signs in an example. He was trying in vain to self-explain the example. A moment later he gave up, saying “Well let’s see if I can just push these in.” He started adapting equations from the example while complaining, “This is called copying too much from the book. I hate that.” This is clearly a case of analogical problem solving of the worst kind. However, after solving the equations and producing a negative formula, he said, “So, according to this, my x-component is equal to -9 , which means, okay. That makes sense. That makes sense. One of these has to always be negative, doesn’t it?” Apparently, the subject figured out why there is a negative sign (although it appears to be based on some kind of symmetry argument, which is an overly general line of mathematical reasoning,

but not the best one for learning this rule). A bit later in the protocol, he failed to use his new knowledge, but corrected his oversight a few lines later (we coded this as a learning event in Table 2). After that, he used the rule fairly consistently. This rather complicated case illustrates how analogy can combine with self-explanation to produce learning via a kind of justified analogy (Kedar-Cabelli, 1985).

Although S101 was one of the Good solvers, he was the worst of the group (Chi & VanLehn, 1992). He self-explained some parts of the examples, but he tended to ignore the details. In particular, he glossed over the equations with the negative signs in them. We suspect that he would have learned more if he had self-explained the examples more carefully as he studied them. His discovery of the negative-sign rule seemed to go much less smoothly than the discoveries of the subjects who were self-explaining during example studying (quoted earlier). Although more evidence is certainly needed before drawing a firm conclusion, it currently appears that self-explanation during example studying might be a more effective than self-explanation in the context of analogical problem solving.

We believe that these cases are just a few of the many ways that analogy can combine with self-explanations, overly general rules, and other learning techniques. The point is, however, that these uses of analogy only briefly interrupt regular problem solving in order to achieve specific meta-goals, such as detecting gaps or filling them. Wholesale analogy, the kind used by the Poor solvers, avoids detecting gaps and thus tends to retard learning. Poor solvers use analogy to achieve base-level goals, such as having a force diagram or having a set of equations. However, the distinction between “meta-goals” and “base-level goals” is notoriously slippery, so the next section tries to formulate a better heuristic for when to use analogy.

5 Discussion

The preceding sections showed that in one study, effective human learners used analogical problem

solving sparingly. With a little bit of computational common sense, we can generalize this result and formulate a heuristic for when such sparse analogical problem solving should be effective.

There are four steps to solving a Newtonian mechanics physics problem:

1. *Define a system.* One must (a) select an idealization of the physical world consisting of idealized bodies that have idealized relationships to other objects and move in idealized trajectories, and (b) decide whether to base the analysis on forces, energies or momenta. In Figure 1, line 1 corresponds to part *a* (albeit, tersely), and part *b* is missing because the text has only introduced one type of analysis (forces) at this point.
2. *Explicate physics quantities.* For each body in the system, one notes the forces, energies or momenta associated with that body. Often, a diagram is drawn to help one remember them. In Figure 1, this occurs during line 2.
3. *Generate equations.* Each body contributes some equations, the connections between bodies contribute other equations, and the problem's boundary conditions may contribute further equations. In Figure 1, the equations are produced on line 6.
4. *Solve the equations for the sought quantity.*

Although logically distinct, these steps are often intermingled in a solver's work. Solvers have no trouble learning this basic procedure. It is often printed in the textbook. It is a specialization of the general 3-step procedure (define a system, formulate a mathematical model, solve it) that is used for all mathematical analysis problems, from lowly arithmetic and algebraic word problems to esoteric branches of science and engineering (see any textbook on systems theory, e.g., Shearer, Murphy & Richardson, 1971).

The system definition step is quite different from the others. There are no real principles for defining a system. Sometimes a man holding a block is treated as one body, and some-

times as two. Sometimes a chain is treated as a single body, sometimes as an infinite sequence of infinitely small bodies, and sometimes as two bodies (cf. Larkin, 1983). Most textbooks emphasize that system definition is more of an art than an algorithm, and few give any heuristics at all for defining systems. Although the problems used in our data are too simple to reveal how the subjects learn about system definition, it seems quite likely that one way that system-defining can be learned is by analogical problem solving and building up a case library that pairs problems with systems.

However, the other 3 steps are governed by well-known principles, such as the force laws (for step 2), Newton's laws (for step 3) and mathematical transformations (for step 4). Moreover, once the system has been defined, the analysis is completely determined. In step 2 (explication of physics quantities), one produces all the forces (or energies or momenta) acting on the system's bodies. In step 3 (equation generation), one produces all the equations implied by those physics quantities.² In step 4, the equations are solved mechanically. The point here is that in physics and many other mathematical analysis task domains, the most important decisions are made during system definition, and the rest of the analysis follows more or less deterministically from those choices. Although analogical problem solving would be useful for learning search control, search control is not very important for steps 2 and 3.

Principles can be used for problem solving, but this does not mean that they necessarily should be used. It may be that case-based reasoning is more effective and/or more efficient than rule-based reasoning. If so, analogical problem solving would be an effective way to master such a task domain. For instance, case-based reasoning might be more effective than principle-based reasoning for certain design tasks (e.g., cooking), in which case a student might be better off practicing wholesale analogical problem solving

² Actually, there are choices to make during step 3 regarding how to rotate the coordinate axes or whether to omit certain equations. These choices affect the difficulty of step 4, but not the ultimate outcome.

rather than rule-based problem solving. However, case-based or analogical problem solving in mathematical analysis task domains often produces only near transfer, whereas principle-based reasoning can produce both near and far transfer (e.g., Reed, 1989; Sweller & Cooper, 1985). Although these studies did not distinguish transfer of system defining knowledge (presumably case-based) from transfer of principles, it nonetheless seems plausible that principle-based reasoning is more effective than case-based reasoning during steps 2 and 3 simply because it is possible to generate problems, such as the ones in Larkin (1983), that are quite dissimilar to textbook problems, thus thwarting or at least complicating case-based reasoning, and yet are amenable to solution by first principles. In short, it seems that the target knowledge for steps 2 and 3 should be principles, and not cases nor search control.

It is worth recalling that our top-level goal is to determine when analogical problem solving is advisable for effective learning. So far, we have argued that analogy should be used during the system defining step and that principles should be used instead of analogy during the other two steps. The next step in the argument is to consider how principles can be learned during steps 2 and 3.

We can safely assume that the learner already knows many principles, so that the learning problem is to detect a gap (missing principle) and fill it, rather than to learn a whole batch of principles at once. In order to detect a gap, one must use principles instead of analogies to achieve goal, for otherwise the body of knowledge containing the gap will not be referenced and detecting the gap would be impossible. Principle-based problem solving will uncover gaps that cause impasses, but not all gaps cause impasses. Thus, it is a good idea to check the intermediate solutions produced by principle-based reasoning, because early detection of an error will facilitate locating the gap that caused it. Analogy is one way to check solutions. By considering the nature of the principle-based reasoning, one can predict when solution-checking is especially important. In physics, missing knowledge

of physics quantities will cause step 2 to produce too few forces, energies, etc., but this will not cause impasses until much later, if at all. Consequently, it is a wise idea to use analogical problem solving to check the results of step 2 before moving on to step 3. This is just what subject S105 did in the case mentioned earlier.³ In general, when the goal is "generate all X that you can think of," where X in S105's case is "forces," then gaps will not cause impasses so it is especially important to check the completeness of the set of generated Xs, and analogy is one way to do that. An experienced learner of mathematical analyses may know this heuristic. There are probably other heuristics about when to check in order to detect errors early via analogy. Several subjects, for instance, routinely checked their equations' signs and trigonometric functions against the examples.

Once a gap is detected, analogy is certainly one possible way to fill it, but it is not easy to predict whether one should use analogy or some other technique, such as instantiating an overly general rule (VanLehn, Jones & Chi, 1991) or explanation pattern (Schank, 1986). A heuristic for this decision would be hard to formulate.

We have arrived finally at our goal, which are heuristics for deciding when to use analogical problem solving. The heuristics are:

1. If the task domain, or some part of the task domain (e.g., steps 2 and 3), has principles, and they are more effective knowledge than cases, and they require little search control, then the target knowledge should be principles.
2. If the target knowledge is principles, they should be acquired by gap filling, which implies:
 - (a) Gap detection: Try to use principles instead of analogies, as a gap may show up as an impasse. Use analogy to check the intermediate results derived

³In order to fill such gaps, Cascade 3 used analogy essentially as a check of step 2, although the implementation was rather baroque (VanLehn & Jones, 1991; in press-a).

via principles, as this may uncover gaps that do not produce impasses.

- (b) Gap filling: Analogical problem solving is one possible technique for filling gaps. Others should be considered as well.

3. If the target knowledge is not principles but cases or search control, then analogical problem solving may be useful.

This conclusion was suggested by human data. There is good evidence that the Poor solvers use analogy "wholesale," as a replacement for principle-based solutions to steps 1, 2 and 3. There is also good evidence that Good solvers avoid analogical problem solving in general. There is some evidence, albeit only a few protocol excerpts, that when Good solvers do use analogy, it is used as an aid to gap filling. In this last section, we have reflected on the human data and, supported by common sense, derived a heuristic for when analogical problem solving should aid learning. This heuristic could have several uses. (1) It helps explain why the Good solvers learned more than the Poor solvers. (2) It is a prescription for effective learning that could perhaps be taught to human students. (3) It could be embedded in a multi-strategy learning.

Clearly, all these implications are testable in their own fashions. A good next step in the research would be to build a multi-strategy learner and experimentally confirm that the heuristic does increase learning. If this succeeds, one could try teaching human students this heuristic, as well as others discovered during the computational experiments, and see if their learning increases as predicted.

Acknowledgements

This research was supported by grant N00014-88-K-0086 from the Cognitive Sciences Division of the Office of Naval Research. We appreciate Joel Martin's and Ted Rees' critiques of the manuscript. This paper also appears in P. Utgoff (Ed.) *Machine Learning: Proceedings of the Tenth International Conference*.

References

- Anderson, J.R. (1990). *The adaptive character of thought*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P. & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.
- Chi, M.T.H. & VanLehn, K. (1991) The content of physics self-explanations. *Journal of the Learning Sciences*, 1(1), 69-106.
- HalLiday, D. & Resnick, R. (1981). *Fundamental of physics, Second edition*. New York: Wiley.
- Kedar-Cabelli, S. (1985). Toward a computational model of purpose-directed analogy. In A. Frieditis (Ed.) *Analogica*. San Mateo, CA: Morgan Kaufman. Reprinted in J. W. Shavlik & T. G. Dietterich (Eds.) *Readings in machine learning*. San Mateo, CA: Morgan Kaufman.
- Larkin, J.H. (1983). The role of problem representation in physics. In D. Gentner & A. Collins (Eds.) *Mental Models*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Reed, S.K. (1989). Constraints on the abstraction of solutions. *Journal of Educational Psychology*, 81, 532-540.
- Schank, R.C. (1986). *Explanation patterns: Understanding mechanically and creatively*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Shearer, J.L., Murphy, A.T. & Richardson, H.H. (1971). *Introduction to systems dynamics*. Reading, MA: Addison-Wesley.
- Siegler, R.S. & Jenkins, E. (1989). *How children discover new strategies*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Sweller, J. & Cooper, G.A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, 2, 59-89.
- VanLehn, K. (1990) *Mind bugs: The origins of procedural misconceptions*. Cambridge, MA: MIT Press.
- VanLehn, K. (1991) Rule acquisition events in the discovery of problem solving strategies. *Cognitive Science*, 15, 1-47.

- VanLehn, K., Jones, R.M. & Chi, M.T.H. (1991). A model of the self-explanation effect. *Journal of the Learning Sciences*, 1, 69-106.
- VanLehn, K. & Jones, R.M. (1991). Learning physics via explanation-based learning of correctness and analogical search control. In L. Birnbaum & G. Collins, (Eds.), *Machine Learning: Proceedings of the eighth international workshop*. San Mateo, CA: Morgan Kaufman.
- VanLehn, K. & Jones, R.M. (1993). Learning by explaining examples to oneself: A computational model. In S. Chipman & A. L. Meyrowitz (Eds.) *Foundations of knowledge acquisition: Cognitive models of complex learning*. Boston: Kluwer Academic Publishing.
- VanLehn, K. & Jones, R.M. (in press-a). Integration of analogical search control and explanation-based learning of correctness. In S. Minton & P. Langley (Eds.) *Machine learning methods for planning and scheduling*. Los Altos, CA: Morgan Kaufman.
- VanLehn, K. & Jones, R.M. (in press-b). What mediates the self-explanation effect? Knowledge gaps, schemas or analogies? To appear in M. Polson (Ed.) *Proceedings of the 15th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.
- VanLehn, K. & Jones, R.M. (in preparation). Self-explanation and analogy.

Multistrategy Learning: An Analytical Approach

Matija Drobnič, Matjaž Gams

Jožef Stefan Institute, Computer Science Dept.

Jamova 39, YU-61000 Ljubljana, Slovenia

Phone: +38 61 159 199, Fax: +38 61 161 029

E-mail: matija.drobnic@ijs.si, matjaz.gams@ijs.si

Abstract

An analytical approach to multistrategy learning is presented. Each single strategy system corresponds to a classifier, and a multistrategy system corresponds to a combined classifier. Therefore, an analytical model of classification with many different classifiers is developed to predict the classification accuracy of the combined classifier. The necessary conditions for the improvement of classification accuracy are determined within the model. The influence of mutual dependence of classifiers is studied in the case of two classifiers. It is shown that the dependence doesn't effect the conditions under which the improvement of classification accuracy emerges. The model is also verified for two different systems learning from medical data. Finally, some special cases are analysed. Our analysis shows that the optimal number of classifiers in

combinations strongly depends on domain and chosen learning algorithms.

Keywords: knowledge representation, machine learning, multiple knowledge, multistrategy learning

1 Introduction

In the case of noisy and incomplete learning data which are typical in real life, artificial intelligence successfully adopted several techniques from other scientific disciplines, e.g. statistics. For example, through empirical measurements it has been shown that statistical estimates are inevitable when constructing or pruning a single decision tree (Breiman *et al.* 1984; Mingers, 1989; Holder, 1991). During the last decade, the question whether to use single-strategy or multistrategy (integrated) learning has often been addressed by many authors.

For example, extensive measurements of many researchers in the field of empirical learning have shown that combining several classifiers results in better results than tuning, no matter how fine, one single classifier (e.g. Buntine, 1990; Clark and Boswell, 1991; Gams, 1989). Basic principles of this approach can be found in the work of several researchers (Brazdil *et al.*, 1991; Handler *et al.*, 1991; Michalski, 1987; Minsky, 1991; Tecucci, 1991). In Bayesian analysis, it has been shown that it is better, at least in general, to design many classifiers and combine them on the basis of probability estimates (Cheeseman, 1991). The results which support this thesis have also been achieved in several related areas, e.g. it has been shown that, when overlapping many filtering methods, the average rating significantly improves with the growing number of methods (Foltz and Dumais, 1992).

In this paper, single-strategy systems correspond to single classifiers, and a multistrategy system corresponds to a combined classifier, the combination of m single classifiers. The presented work attempts to enhance the grounding of multistrategy learning. Namely, the proposed models are expected to predict whether

$$X = \{V_{11}, \dots, V_{1M_1}\} \times \dots \times \{V_{N1}, \dots, V_{NM_N}\}. \quad (2.1)$$

Let X be a measurement space and C a set of all possible classes. A domain is then a set of ordered pairs

$$D \subseteq \{(\bar{x}, c): \bar{x} \in X, c \in C\}, \quad (2.2)$$

and a classifier d is a function

$$d: X \rightarrow C \times \mathbb{R} \quad (2.3)$$

$$\bar{x} \mapsto (d(\bar{x}), c, d(\bar{x}).cf)$$

the multistrategy approach enables the improvements in the given case or not. In our approach, an m^2 -parametric model is used to estimate the classification accuracy of a combination of m classifiers and to compare it to the most accurate among them. In the special case of two classifiers, the effect of mutual dependence is introduced using a hybrid classifier. The comparison of two classification methods on medical domain is performed and the results are in good agreement with the predictions of our model. For the analysis of behavior of more classifiers, our model is simplified to depend on two parameters only, so that we can study the influence of the properties of single classifier on the performance of combination.

2 Classification With m Classifiers

2.1 General analysis

In the following, examples are described in attribute-value languages. Each example belongs to exactly one class. Let us denote N attributes as A_1, \dots, A_N and their values as $(V_{11}, \dots, V_{1M_1}), \dots, (V_{N1}, \dots, V_{NM_N})$. The measurement space is then defined as

Here, \mathbb{R} are real numbers, c denotes the class of object \bar{x} and cf is a confidence factor of classification.

Let us now consider a set of classifiers, all mapping from the same X to the same C , $M = \{d_1, d_2, \dots, d_n\}$. There are different ways of combining the classification of

these classifiers. One can take the result of classification with the greatest confidence factor (best-one principle). We can also take the class that was proposed by the majority of classifiers (majority principle). In addition, this voting can also

$$d_M : X \rightarrow C \times \mathfrak{R}$$

$$\bar{x} \mapsto (d_i(\bar{x}).c, d_i(\bar{x}).cf); d_i(\bar{x}).cf = \max_{d_j \in M} \{d_j(\bar{x}).cf\} \quad (2.4)$$

When classifying a single vector $\bar{x} \in X$ with classifiers from M , each of them can predict either correct (success, T) or wrong class (failure, F). The result of the multiple classification can be described by a situation vector $\bar{s} \in S$, where $S = \{T, F\}^m$ and $\|M\| = m$. In this notation, we set

$$s_i = \begin{cases} T, & (\bar{x}, d_i(\bar{x}).c) \in D \\ F, & (\bar{x}, d_i(\bar{x}).c) \notin D \end{cases}$$

Let us now determine the probability of the correct classification of m combined independent classifiers. For each situation

$$p_{\bar{s}} = \left(\prod_{i, s_i=T} p_i \right) \left(\prod_{j, s_j=F} (1 - p_j) \right). \quad (2.5)$$

The determination of $q_{\bar{s}}$ is, however, rather complicated and depends on the combining technique. Since we use the best-one principle, i.e. the result of multiple classification is the class predicted by the classifier with the greatest confidence factor, we have to introduce parameters q_{ij} in our model to denote the probability that the confidence factor of the i -th classifier is greater than the confidence factor of the j -th classifier in a situation where

be weighted. In this presentation, we are dealing with the best-one principle. A *multiple or combined classifier* on a set M , based on the best-one principle, is therefore defined as

$\bar{s} \in S$, we have to determine the probability $p_{\bar{s}}$ of its occurrence and the probability of the correct classification in a given situation $q_{\bar{s}}$.

In a given domain D , the probability of correct classification of the i -th classifier is denoted by p_i . These probabilities play major role in our analysis. We will assume $0 < p_i < 1$ to avoid trivial results. Under the assumption that all the classifiers are mutually independent, the probability of the occurrence of the situation $\bar{s} \in S$ is

the first one succeeds and the second one fails

$$q_{ij} = P(d_i(\bar{x}).cf > d_j(\bar{x}).cf | s_i = T, s_j = F). \quad (2.6)$$

Now we have to determine the probability, that at least one of the classifiers $d_i, \bar{s}_i = T$ has its confidence factor greater than all the classifiers $d_j, \bar{s}_j = F$

$$q_{\bar{s}} = P \left(\bigvee_{i, s_i=T} \left[\bigwedge_{j, s_j=F} (d_i(\bar{x}).cf > d_j(\bar{x}).cf | s_i = T, s_j = F) \right] \right). \quad (2.7)$$

Let us first express the inner factor

$$q_{i\bar{s}} = \left[\bigwedge_{j, s_j = F} (d_i(\bar{x}).cf > d_j(\bar{x}).cf | s_i = T, s_j = F) \right] \quad (2.8)$$

$$= \prod_{j, s_j = F} q_{ij},$$

where q_{ij} are defined in (2.6). In a special case where $s_i = T \forall i$, we define $q_{i\bar{s}} = 1$. Let $n_{\bar{s}}$ be the number of T elements in \bar{s} , $n_{\bar{s}} = \sum_{i, s_i = T} 1$. Now we can write the general expression for $q_{\bar{s}}$

$$q_{\bar{s}} = \begin{cases} 1 - \prod_{i=1}^{n_{\bar{s}}} (1 - q_{i\bar{s}}), & n_{\bar{s}} > 0 \\ 0, & n_{\bar{s}} = 0. \end{cases} \quad (2.9)$$

The probability of correct classification of a multiple classifier on a set M is obtained as a sum over all possible situations

$$p_M = \sum_{\bar{s} \in S} p_{\bar{s}} q_{\bar{s}}. \quad (2.10)$$

This sum is the basis of further analysis.

2.2 Special cases

Let us now take a look at some special cases. In the case of 2 independent clas-

sifiers, the 4 possible situations are presented in Table 1.

\bar{s}	$p_{\bar{s}}$	$q_{\bar{s}}$
(F, F)	$(1 - p_1)(1 - p_2)$	0
(T, F)	$p_1(1 - p_2)$	q_{12}
(F, T)	$p_2(1 - p_1)$	q_{21}
(T, T)	$p_1 p_2$	1

Table 1: 4 possible situations for 2 independent classifiers

For the probability of a correct classification of a multiple classifier we obtain

$$p_M = \sum_{\bar{s} \in S} p_{\bar{s}} q_{\bar{s}} \\ = p_1(1 - p_2)q_{12} + p_2(1 - p_1)q_{21} \\ + p_1 p_2. \quad (2.11)$$

For a set with 3 independent classifier, we obtain 8 different situations, which are shown in Table 2.

\bar{s}	$p_{\bar{s}}$	$q_{\bar{s}}$
(F, F, F)	$(1 - p_1)(1 - p_2)(1 - p_3)$	0
(T, F, F)	$p_1(1 - p_2)(1 - p_3)$	$q_{12}q_{13}$
(F, T, F)	$p_2(1 - p_1)(1 - p_3)$	$q_{21}q_{23}$
(F, F, T)	$p_3(1 - p_1)(1 - p_2)$	$q_{31}q_{32}$
(T, T, F)	$p_1 p_2(1 - p_3)$	$q_{13} + q_{23} - q_{13}q_{23}$
(T, F, T)	$p_1 p_3(1 - p_2)$	$q_{12} + q_{32} - q_{12}q_{32}$
(F, T, T)	$p_2 p_3(1 - p_1)$	$q_{21} + q_{31} - q_{21}q_{31}$
(T, T, T)	$p_1 p_2 p_3$	1

Table 2: Overview of situations for 3 classifiers

In our model, each set M of m classifiers on a given domain is described by m^2 parameters, which can be arranged into a matrix $m \times m$

$$A_m = \begin{pmatrix} p_1 & q_{12} & \dots & q_{1m} \\ q_{21} & p_2 & \dots & q_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \dots & p_m \end{pmatrix}. \quad (2.12)$$

The upper data description has been used for all our analyses.

2.3 Mutual dependence of classifiers

The assumption that classifiers are mutually independent, is rarely met when

dealing with real-life domains. The question is, how relevant are the results of our model in case of dependent classifiers. Let us again assume that d_1 and d_2 are two independent classifiers and their probabilities of correct classification are p_1 and p_2 , respectively. A hybrid classifier d_2^* is then a function which gives the same result as d_1 with probability d and the same result as d_2 with probability $1 - d$. Then we can form a multiple classifier with d_1 and d_2^* . The results are presented in Table 3.

\bar{s}	$p_{\bar{s}}$	$p_{\bar{s}}, d = 0$	$p_{\bar{s}}, d = 1$	$q_{\bar{s}}$
(F, F)	$(1 - p_1)((1 - d)(1 - p_2) + d)$	$(1 - p_1)(1 - p_2)$	$1 - p_1$	0
(T, F)	$p_1(1 - d)(1 - p_2)$	$p_1(1 - p_2)$	0	q_{12}
(F, T)	$(1 - p_1)(1 - d)p_2$	$(1 - p_1)p_2$	0	q_{21}
(T, T)	$p_1((1 - d)p_2 + d)$	p_1p_2	p_1	1

Table 3: 4 possible situations for 2 dependent classifiers

As a sum over all situations, we obtain

$$\begin{aligned} p_M^* &= \sum_{\bar{s} \in S} p_{\bar{s}} q_{\bar{s}} \\ &= (1 - d)p_1(1 - p_2)q_{12} \\ &\quad + (1 - d)p_2(1 - p_1)q_{21} \\ &\quad + (1 - d)p_1p_2 + dp_1 \\ &= (1 - d)p_M + dp_1. \end{aligned} \quad (2.13)$$

For $d = 0$, the result is obviously the same as for two independent classifiers. For $d = 1$, the classifiers are equal and the obtained result is equal to the result of one classifier. For $0 < d < 1$, the classification accuracy p_M^* lays between p_1 and p_M . If we assume, without loss of generality, that $p_1 > p_2$, it holds $p_M > p_1 \Rightarrow p_M^* > p_1$.

Though the dependence shrinks the gain of classification accuracy, it doesn't effect the conditions under which the gain is obtained. Similar result can also be shown for relative gain of classification accuracy

$$\begin{aligned} r_M^* &= \frac{p_M^* - p_1}{p_1} \\ &= \frac{(1 - d)p_M + dp_1 - p_1}{p_1} \\ &= (1 - d) \frac{p_M - p_1}{p_1} \\ &= (1 - d)r_M. \end{aligned} \quad (2.14)$$

It is obvious, that the mutual dependence does not effect the sign of r_M .

3 Measurements In Medical Domain

3.1 Description of data and classifiers

For our measurements, real-life data on patients and their coronary disease diagnoses have been used. Every patient was described by 30 attributes and fell into one of the three possible classes. The set contained 112 patients and was ten times randomly partitioned into a training (80 patients) and a testing set (32 patients). The two chosen methods were naive Bayes using an m-estimate ($m=2$) for probabilities as the first method and the k-th nearest neighbor ($k=5$) as the second

one. Neither of methods belongs to the AI field, but they are often used when comparing classification accuracies of different AI methods as references. They both return the class distribution for a given example. In our experiments, the major class was returned as a result of the classification, and its probability as a confidence factor.

3.2 Verification of the model

During our experiments, we have measured the classification accuracies p_i , the probabilities of occurrences of all situations $p_{\bar{s}}$ as well as probabilities of correct classifications in given situation $q_{\bar{s}}$. The results are presented in Table 4.

	p_i		$p_{\bar{s}}$				$q_{\bar{s}}$	
		p_2^*	$p_{(F, F)}$	$p_{(T, F)}$	$p_{(F, T)}$	$p_{(T, T)}$	$q_{(T, F)}$	$q_{(F, T)}$
0	0.813	0.844	0.156	0.000	0.031	0.813	?	0.000
1	0.875	0.813	0.125	0.063	0.000	0.813	1.000	?
2	0.844	0.844	0.125	0.031	0.031	0.813	1.000	1.000
3	0.781	0.750	0.219	0.031	0.000	0.750	0.000	?
4	0.875	0.844	0.094	0.063	0.031	0.813	1.000	0.000
5	0.813	0.844	0.156	0.000	0.031	0.813	?	0.000
6	0.844	0.875	0.125	0.000	0.031	0.844	?	0.000
7	0.875	0.906	0.094	0.000	0.031	0.875	?	0.000
8	0.938	0.969	0.031	0.000	0.031	0.938	?	1.000
9	0.844	0.844	0.094	0.063	0.063	0.781	1.000	0.000
\bar{x}	0.850	0.853	0.122	0.025	0.028	0.825	0.800	0.250

Table 4: Measured probabilities p_i , $p_{\bar{s}}$ and $q_{\bar{s}}$

In case that a certain situation \bar{s} doesn't occur at all, the corresponding conditional probability $q_{\bar{s}}$ cannot be meaningfully estimated and is therefore denoted by "?". From Table 4 we can see that the classification accuracies p_1 and p_2^* differ on aver-

age about 0.3%. Both classifiers are quite similar and they also use the same data for learning. We can also see that both classifier rarely disagree ($p_{(T, F)} + p_{(F, T)} < 6\%$). This fact indicates a high level of mutual dependence of classifiers.

In order to estimate the dependence coefficient d and the classification accuracy of the virtual independent classifier p_2 , we fit the situation probabilities $p_{\bar{s}}$ from Table 3 and the accuracy of the hybrid classifier $p_2^* = dp_1 + (1 - d)p_2$ to the up-

per measurements. The resulting values of d and p_2 were then used for evaluation of (2.13). The predictions of (2.13) ($p_M(calc.)$) are compared to the measured classification accuracy of combined classifier ($p_M(meas.)$) in Table 5.

	d	p_2	$p_M(meas.)$	$p_M(calc.)$
0	0.834	1.000	0.813	0.813
1	0.928	0.013	0.875	0.875
2	0.765	0.844	0.875	0.875
3	0.958	0.051	0.750	0.750
4	0.680	0.777	0.875	0.875
5	0.834	1.000	0.813	0.813
6	0.801	1.000	0.844	0.844
7	0.752	1.000	0.875	0.875
8	0.500	1.000	0.969	0.969
9	0.524	0.843	0.844	0.844
\bar{x}	0.784	0.864	0.853	0.853

Table 5: Dependence d and comparison of model and measurements

We can see that the dependence is very high (over 75% in average) which doesn't promise a substantial gain of classification accuracy. In two cases (partitions 1 and 3), the fitting procedure chose a very small number for p_1 and tried to compensate this with larger value for d . For other partitions, however, the obtained values of d seem quite realistic.

The gain of classification accuracy with respect to p_1 and p_2^* is obtained only for partition 2 (compare Table 4 and Table 5). Why? Let us assume that both classification accuracies are equal. $p_1 = p_2 = p$. Under this assumption, the equation

(2.11) is

$$\begin{aligned}
 p_M &= \sum_{\bar{s} \in S} p_{\bar{s}} q_{\bar{s}} \\
 &= p(1 - p)(q_{12} + q_{21}) + p^2.
 \end{aligned}$$

The necessary condition for relative gain is then

$$\begin{aligned}
 \frac{p_M - p}{p} &> 0 \\
 (1 - p)(q_{12} + q_{21}) + p - 1 &> 0 \\
 (1 - p)(q_{12} + q_{21} - 1) &> 0 \\
 q_{12} + q_{21} &> 1
 \end{aligned}$$

Indeed, this condition is met only for partition 2. The correlation between the correct classification and confidence factor is certainly a vital condition for the success of a combined classifier.

A closer look at the Table 4 explains the reason for the failure: the first classifier (naive Bayes) seems to be too self-confident as we can see from the values of $q_{\bar{s}}$. Even if it fails, its confidence factor is greater than the one of the second classifier (k -th nearest neighbor). Therefore, the simple best-one mechanism doesn't behave fairly and the failure of the first classifier easily superseeds the success of the second one. It is important to notice, that the gain of classification accuracy can be expected only for more balanced values of $q_{\bar{s}}$.

4 A Simplified Model Of m Classifiers

For the analysis of behavior of more classifiers, our model is rather hard to use, since it describes a system of m classifiers with m^2 parameters. Therefore we have to make a sound simplification to study the influence of the properties of a single classifier on the performance of combination for $m > 2$. First, we assumed

that all classification accuracies p_i are the same. We base this assumption on the observation that in real-life domains (also in our case) the accuracies of different classifiers do not differ very much. Second, we also assumed that all the conditional probabilities of successful multiple classification q_{ij} are the same. This is definitely not true in our case. However, we claim that one of the reasons for this is a relative small number of situations where the classifiers disagree and our results for q_{ij} aren't significant. Furthermore, as mentioned above, these parameters should be more balanced to obtain a fair behavior of the best-one combining method. And finally, it seems that, in situations of practical interest for the use of multistrategy learning, these parameters would depend more on the domain than on the classifying methods.

Under the above assumptions, the equations (2.10) gives us the following results for relative gains ($p > 0$):

$$m = 2; \quad r_M = (p - 1)(1 - 2q)$$

$$m = 3; \quad r_M = (p - 1)(1 + p - 6pq - 3q^2 + 6pq^2)$$

$$m = 4; \quad r_M = (p - 1)(1 + p + p^2 - 12p^2q - 12pq^2 + 24p^2q^2 - 4q^3 + 8pq^3 - 8p^2q^3 + 6pq^4 - 6p^2q^4)$$

$$m = 5; \quad r_M = (p - 1)(1 + p + p^2 + p^3 - 20p^3q - 30p^2q^2 + 60p^3q^2 - 20pq^3 + 40p^2q^3 - 40p^3q^3 - 5q^4 + 15pq^4 + 15p^2q^4 - 20p^3q^4 + 10pq^6 - 30p^2q^6 + 20p^3q^6)$$

Since we simplified our model to depend on two parameters only, the results can be presented in 3D space. The upper functions are shown in Figure 1. Only where

they are positive, the gain of classification accuracy can be expected.

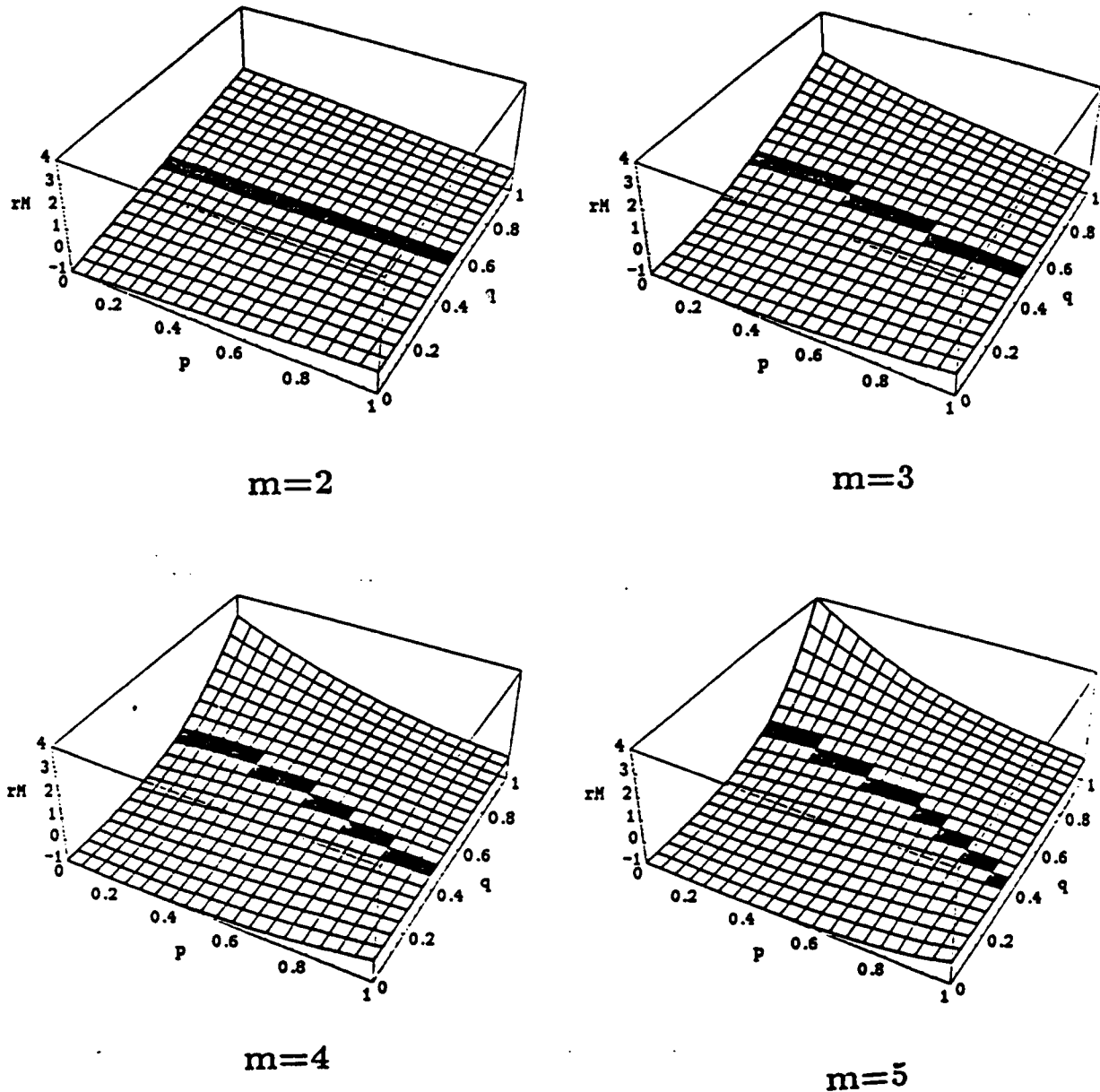


Figure 1: Dependence of r_M on p and q

The line in Figure 1 indicates the border between the areas of positive and negative gain. We can clearly see that the improvement of classification accuracy cannot be expected for small values of q . Also, the relative improvement is obviously greater for smaller values of p .

It seems that the relative gains for $q = 1$ and very small classification accuracy grows linearly with the number of classifiers in combination m . Let us again examine (2.10). If q is set to 1, all the q_i except the first one are also equal to 1.

In the limit $p \rightarrow 0$, only the terms $p_{\bar{s}}$ of the form $p + \dots$ are of interest. Since only the probabilities of situations with one T component are of this form and there are m such situations (not taking into account the situation without T components, where $q_{\bar{s}} = 0$), the result is

$$p_M = m p + \dots,$$

as we expected.

5 Conclusions

The analysis of multistrategy learning in the form of multiple classification has been presented. An analytical model has been developed to predict the classification accuracy of the combination of different classifier. The analytical results were compared to the measurements of classification of the methods on real-life domain. The average error of the model prediction was under 1%, so it seems that we can rely on it if we want to determine whether to use the multistrategy learning or not in the given situations.

The assumption of mutual independence of different classifiers is rarely met in real life, since all the classifiers typically use the same set of examples for learning. Therefore we introduced the dependence parameter d in our model to analyse the influence of mutual dependence on the classification accuracy of the multiple classifier at least in the case of two classifiers. We showed that the dependence shrinks the expected gain (positive or negative) of classification accuracy by factor $(1 - d)$. However, it does not effect the conditions under which we expect the improvement of classification accuracy. We also showed, how to fit our

model to the measured probabilities to estimate the value of d for the given classifiers.

Within the simplified model it has been shown that the gain of the classification accuracy grows linearly with the number of combined classifiers if their accuracies are small and the probabilities for the success of combination are high. However, when the accuracies of single classifiers are high, the expected gain decreases and blindly adding new classifiers into the combination doesn't seem to be the right method.

Our work presents new indications that the current learning techniques can be substantially improved by the use of multiple knowledge. However, it also indicates that the improvement appears only under certain conditions, which have to be kept in mind during the development and implementation of new combining methods.

References

- Breiman, L., Friedman, J., H., Olshen, R., A., Stone, C., J., "Classification and Regression Trees," *Wadsworth International Group*, 1984.
- Brazdil, P., Gams, M., Sian, L., Torgo, L., van de Velde, W., "Learning in Distributed Systems and Multi-Agent Environments," *Proc. of EWSL-91*, Porto, Portugal, 1991.
- Buntine, W., "A Theory of Learning Classification Rules," *Ph.D. thesis*, Sydney, Australia, 1990.
- Cheeseman, P., On Finding the Most Probable Model, in *Computational Models of Scientific Discovery and Theory Formation*, Shrager, J. and Langley, P., (Eds.) Morgan Kaufmann, 1991.

Clark, P. and Boswell, R., "Rule Induction with CN2: Some Recent Improvements," *Proc. of EWSL-91*, Porto, Portugal, 1991.

Foltz, P.W. and Dumais S.T., "Personalized Information Delivery: An Analysis of Information Filtering Methods," *Communications of the ACM*, Vol. 35, 12, 1992.

Gams, M., "New Measurements Highlight the Importance of Redundant Knowledge," *Proc. of EWSL-89*, Montpellier, France, 1989.

Handler, J., Bobrow, D., Gasser, L., Hewit, C. and Minsky, M., "Multiple Approaches to Multiple Agent Problem Solving," *Proceedings of IJCAI'91*, Sydney, Australia, 1991.

Holder, L.B., "Maintaining the Utility of

Learned Knowledge Using Model-Based Adaptive Control", *Ph.D. thesis*, University of Illinois, USA, 1991.

Michalski, R.S., "Machine Learning Tutorial", *IJCAI'87*, Milan, Italy, 1987.

Mingers, J., "An Empirical Comparison of Pruning Methods for Decision Tree Induction", *Machine Learning*, Vol. 4, 2, 1989.

Minsky, M., "Logical versus Analogical or Symbolic versus Connectionist or Neat versus Scruffy", *AI magazine*, Vol. 12, 2, 1991.

Tecucci, G., "Learning as Understanding the External World", *Proc. of the First International Workshop on Multistrategy Learning*, Harpers Ferry, USA, 1991.

Reflection and Analogy in Memory-based Learning

Enric Plaza Josep Lluís Arcos
Institut d'Investigació en Intel·ligència Artificial, CSIC
 Camí de Santa Bàrbara, 17300 Blanes, Catalunya, Spain.
 {plaza | arcos}@ceab.es

Abstract

In this paper we will discuss the significance of memory and reflection in integrated learning architectures. The Massive Memory Architecture, a uniform architecture based on episodic memory and case-based reasoning, is described. Its reflective capabilities are described and we put forth the hypothesis that learning methods are inference methods with reflective capabilities, i.e. methods requiring a self-model of the system. Self-models and method implementation are based on conceptual, knowledge-level descriptions of inference. We show how the MMA reflective capabilities can be used for integrating learning and problem solving.

1 Introduction

In this paper we will discuss the significance of memory and reflection in integrated learning architectures. The role of memory in learning systems is widely recognized, as by Michalski's "equation": $\text{learning} = \text{inference} + \text{memory}$. This equation is a summary of the basic abilities a learning system has to have in order to be able to learn: the reasoning ability and the memory storage and retrieval ability. In the next section we will introduce the Massive Memory Architecture (MMA), an integrated learning architecture based on episodic memory we are developing. We claim there is a second crucial issue in learning architectures that is not so widely recognized, namely that of *reflection*. In order to justify this approach we just have to realize that a system that has to learn of its own experience has to be able to inspect its own behav-

ior (that has to be represented and stored in its memory), analyze it and discover what aspect is responsible for a failure (or a success, or a delay, etc) and decide how to transform itself (its knowledge, procedures for decision, etc) so that its future behavior is to be considered more appropriate (or adapted). All this is fairly general, but the important think is the necessity of the system to be able to self-inspection and self-modification, i.e. the capability of *reflection*.

Reflection capabilities have been an issue with long tradition in mathematics, philosophy, logic, linguistics and computer science. These capabilities can be formally studied since Feferman's (1962) work on reflection principles and computational approaches exist since FOL (Weyhrauch 89) and 3-LISP (Smith 85). For our purposes, reflection can be used to think of learning methods as a kind of inference or reasoning able to introspect into the systems representation of its behavior and modify the system structure and future behavior in an appropriate way. Therefore we will consider learning as an *inference of reflective nature that uses a self-model of the system*. This reflective nature means that learning is a system's component able to self-inspect and self-modify the system itself. In order to infer new decisions from the results and behavior of other inference processes, those results and behavior have to be represented and stored in the memory for the learning inference to be able to work with them.

A third characteristic is the scheme of representation used for memory and inference. We do not follow logic-oriented formalisms, but

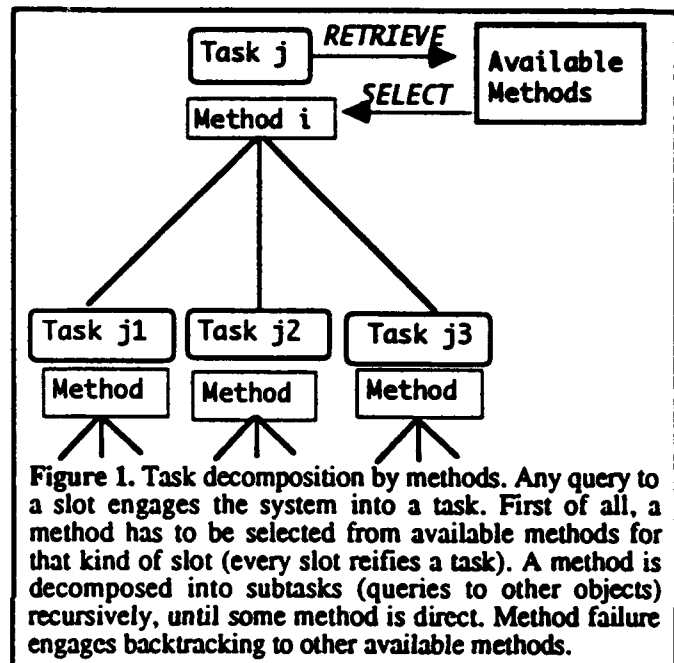
the conceptual or "knowledge-level" frameworks, like KADS (Wielinga 92) or Commet (Steels 90) developed for analyzing and representing complex reasoning in expert systems. In these frameworks reasoning is represented in terms of tasks (or goals), methods that may achieve them, the subtasks needed to realize methods, and the knowledge or models used by those methods. As we will see, this approach allows us (1) to analyze learning methods as a form of complex reasoning based on task/method decomposition, and (2) represent and implement learning methods in a uniform way using this task/method decomposition and (3) integrate them in a uniform architecture using reflection principles to relate learning with problem solving.

2 The Massive Memory Architecture

The MMA is an experimental framework for experience-based learning and reasoning. It is based on memorisation of past episodes of problem solving and in a default behavior that resorts to analogous past cases (precedents) to solve new situations. This is a default behavior in the sense that it is used when no concrete domain knowledge is available. The analogical inference is modelled as an inference pattern Retrieve/Select/Adapt. This pattern is reified into an analogical inference method object, where different retrieve or select methods can be used that are domain dependent. The fact that inference methods are first class objects means that inference methods can be programmed also. Inference methods in MMA are methods that follow a Retrieve / Select / Adapt pattern. Thus, an inference method is a reification of the basic inference pattern of the architecture. Analogical methods are inference methods that follow a Retrieve by similarity methods and then may have Select methods also of similarity or using domain-based, knowledge-intensive methods. Inheritance is also represented and implemented by explicit inference methods that use a retrieve method that follows a link (e.g. the type link, but other inheritance methods are used, like the species link). Thus, analogy and inheritance are integrated as patterns of search in memory.

Every episode of problem solving of MMA is represented and stored as an episode in memory. This is the main point of the reification

process: create the objects that can be usable for learning and improving future behavior. MMA records memories of successes and failures of using methods for solving tasks. Since inference methods are also methods, learning can be applied to different types of retrieval methods and selection methods used in the process of searching and selecting sources of knowledge in memory.



Analogical methods are inference methods that follow a task decomposition of Retrieve / Select / Adapt. Since different methods can be used for these subtasks, multiple methods of case based reasoning can be integrated. The characteristic of analogical methods is that the Retrieve method uses a similarity-based method. Select methods can also be based on similarity or can be domain-based, knowledge-intensive methods. All inference methods are such because they are able to search for sources from which some knowledge may be retrieved. The types of knowledge retrieved is either domain knowledge (as methods) and experiential knowledge (situations of failure and success). Experiential knowledge is used by MMA to bias the preferences of future actions using precedent cases stored in past episodes. The uniform nature of MMA (all representation is in the form of slots in objects) supports learning at all decision points of the system.

2.2 Elements of inference

Our purpose is then to reify the problem solving process into a collection of abstract inference components. We call this *inference-level reflection*. We have developed NOOS, a frame-based language with reflective capabilities to implement the Massive Memory Architecture. In NOOS, those elements are *tasks* (or goals) and *methods* (or ways of achieving a goal), and *theories*. Therefore, all problem solving in a domain will be by means of a task to be solved and the methods that can be relevant to solve it. Moreover, if a method does not directly solve a task, it may induce subtasks that need to be solved. The NOOS approach is uniform, and this entails that the problem solving process is also described in the system in terms of tasks and methods. For instance, if there is no method specified for solving a given task, the *task* of the problem solving process is to find such a method; or if there are more than one method that can possibly solve a task, a *task* of problem solving process is to choose among them. A way to do it is trying them out until one works: that would be a *method* for such a *task*. A task is engaged when exists a query asking the filler of a slot, expressed as (>> F of U) in NOOS syntax and F(U) in abstract syntax.

The third component of our framework for inference-level reflection are *theories*. Knowledge cannot be simply represented as an unorganised bag of axioms lest the language cancels its capability of manipulating different theories. In NOOS, every theory is reified into an object of the language. For instance, the person-theory object reifies the theory of what we know about persons, and we can have another theory of what we know is typically true for persons reified in the typical-person-theory object. Certainly, that leaves open the issue of how these two theories relate to each other, i.e. how to use and manipulate them appropriately. In the uniform approach of NOOS, there is an *inference theory* that specifies how these domain theories are treated when trying to solve some problem about persons. Relations among theories can be stated in the usual way, as in the assertion

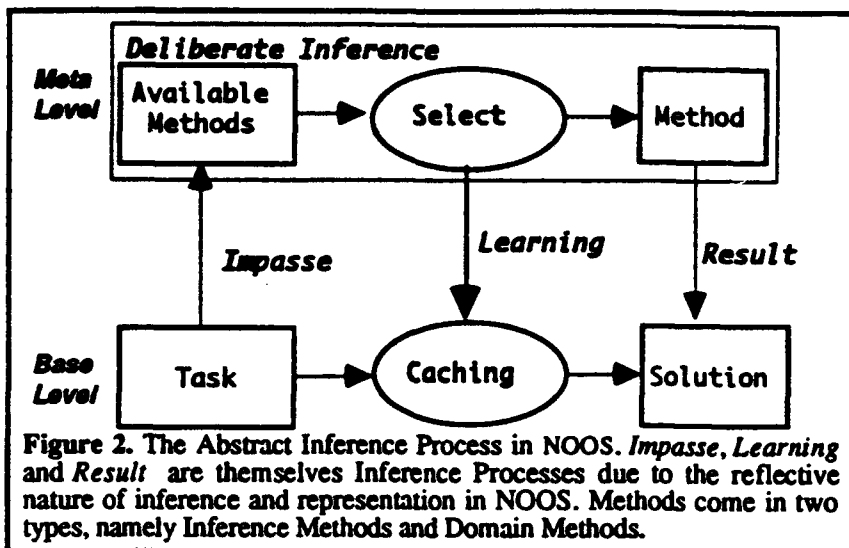
Default-Theory(person-theory) =
= typical-person-theory

where we state that the typical-person-theory contains what we know by default about persons. Moreover, an inference theory may say, e.g. that in order to infer some proposition P(John) we should use a default theory typical-person-theory only after not being able to infer P(John) using the main theory person-theory, where the Default-Theory relation is used to prefer one theory before the other according to the current situation. This knowledge is contained in the person-inference-theory object.

The basic inference process of NOOS follows the Retrieve/Select/Adapt pattern. The other notion needed to explain the inference process are *impasses*. When a query for a slot (>> father of John) is evaluated a new task is started. Then either

- (i) task "father(John)" has a method like (>> husband mother of self), or
- (ii) a no-method impasse occurs.

Case (i) is called *spontaneous inference* and occurs at the base level. However, in (ii) the impasse causes NOOS to search at a meta-level for possible methods to use. Impasses are handled by metaobjects, that is to say MMA is an impasse-driven reflective architecture. The architecture specifies which types impasses can appear, and which kind of metaobject will handle them. The no-method impasse in a slot F(U) is handled by its metafunction. There the applicable methods can be retrieved and selected (maybe trying them out) and the solution is cached in the slot (see Fig. 2). Every impasse is an opportunity for learning and the reification process creates and stores the objects needed to represent the situation (so that it can be useful in the future). In the slot example, the information stored is the successful method and the methods tried that failed. The inference can be more complex, e.g. maybe the applicable methods for slot query (>> father of John) are unknown. This is a new kind of impasse: the no-metafunction impasse and is handled by the metatheory of John that possesses inference methods able to search, retrieve and select methods in other objects. The point to notice is that the uniformity of NOOS treats all situations in the same way. As in Soar, every impasse arises from lack of knowledge: either because the system does not know what to do, or it has several possibilities to act and has to decide



among them. The first type of impasses is handled by inference methods that know how to retrieve sources of knowledge. Multiple possibilities are handled by strategic clichés, objects that know about preferencing and selecting among choices.

3 Reflection and self-models

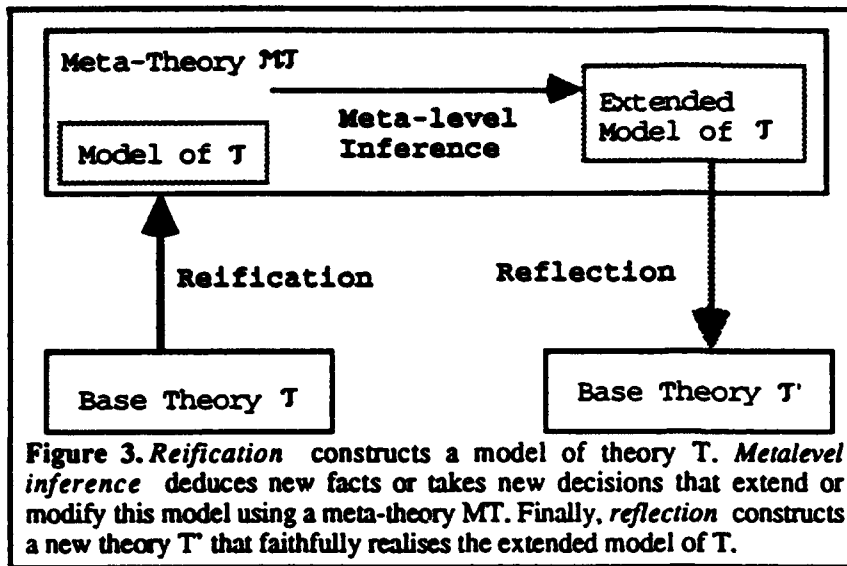
We will first justify the claim that learning is a type of meta-level inference, arguing that learning requires a self-model of the system. Then, we will explain the processes of reification and reflection. Self-models are required because of the integration of learning methods with a problem solving system. In general, a learning method has to have a model of *what are* "successes" and "failures" in the architecture, and of other relevant concepts for learning (e. g. the SOLE-ALTERNATIVE concept in EBL-PRODIGY). These concepts are part of the learning *self-model* of the architecture. These models depend upon what is needed by the method, i.e. they are different models for different learning methods (this is called "white-box requirement" in (Carbonell 91), meaning that any ML method has to be able to view and represent what it requires of the problem solver). Moreover, the learning method needs to be able to effectively inspect part of the structure and behavior (state) of the architecture, and interpret that into its method-specific model. Therefore, *learning can be viewed a type of meta-level inference*. A meta-level inference is a kind of inference able to inspect (to have a model of) the base-level, in-

fer some new decision, and modify the base-level in such a way that it complies to that decision (Smith 85). The Massive Memory Architecture has a model of the usage of methods for solving tasks. MMA self-model for case-based reasoning consists of the decisions, successes and failures of methods that are declaratively recorded in the system's memory of cases. This model is reified in MMA, i.e. those concepts are computational objects accessible to the system. This model allows MMA to retrieve methods already used for similar problems and to control search by analogical

transfer of past decisions in similar situations to the current problem. The meta-level issue is also implicit in ILT, the inferential learning theory (Michalski 91). In ILT learning methods are analysed as higher-level inference patterns the result of which are "knowledge transmutations", i.e. the modification of the system's knowledge as mandated by the inference performed by the learning method.

3.1 Reification and reflection

The *reflection principles* specify the relationship between a theory T and its meta-theory MT . The *upward principles* specify the reification process that encodes some aspects of T into ground facts of MT . That is to say, reification constructs a particular model of T in the language used by MT . The nature of reification and the model constructed is open, i.e. it depends on the purpose for which the coding is made. We will use in MMA a knowledge-level model of task/method/theory decomposition (explained in §2) as a meta-model of the base-level inference. We follow a framework similar to the Components of Expertise (Steels 90), A similar approach is taken in (Akkermans 92) where the meta-model is the KADS modelling framework (Wielinga 92) for expert systems. However, we do not follow them strictly, except in the general idea of using as "elements of inference" goals, methods, and theories. The meta-theory MT contains knowledge that allows to deduce how to extend this model deducing new facts about it. This deduction process is called meta-level in-



ference, and the content of this theory is again specific to the purpose at hand (the meta-theory is indeed no more than a theory). Finally, *downward principles* specify the reflection process that given a new, extended model of *T* has to transform the theory *T* to a new theory *T'* that complies to that new model. A more detailed explanation of the reflective principles and of the semantics of NOOS can be found in (Plaza 92a).

3.2 Self-models in MMA

Our hypothesis is that different types of learning methods would require different self-models of the architecture. The current implementation of MMA has a model of the methods used *for each task*: methods that have been proposed (by an inference method), methods that have been tried but failed, and the method that has succeeded. This information is stored in an object called *slot-access*. In the following we will use quotes "X" to designate the reification of X.

```

Access-Name("Age(John)") => "Age"
Domain("Age(John)")    => #<John>
Method("Age(John)")    => #<Method Age-method-3>
Failed("Age(John)")    => #<Method Age-method-5>
Referent("Age(John)")  => #<32-Years>
  
```

This self-model is used by inference methods to retrieve and transfer the metafunction (containing the available methods) from a task solved into a precedent case to a task in the present problem, and for inferring preferences over method selection based on their success or failure in those precedents. For instance,

the MMA can obtain the method that successfully computed the age of John using this query:

```

(>> method reify (>> age of John))
=> #<Method Age-method-3>
  
```

Other learning methods that we are incorporating to MMA use this self-model but also require its extension. This is as expected because of our self-models hypothesis implies that every learning method may need to know different aspects of the architecture. We are then in a process where an analysis of those learning methods elucidate which aspects of NOOS that are hidden or internal to its implementation are to be reified and made accessible to the architecture.

4 A diagnosis example

Let us show the dynamics of the MMA with a simple example like the car does-not-start diagnosis. *johns-car* gives the problem data where complaint is that the car does not start and the task is diagnosis.

```

(define johns-car
  (owner john)
  (complaint does-not-start)
  (gas-level full)
  (battery-voltage low-voltage))
  
```

We may have two ways of solve the problem, one is the knowledge-based diagnosis using a generate and test method, and the other one is a precedent-based method. The knowledge to diagnose cars is in *car-g&t-diagnosis* but it might be incomplete (e.g. not always can generate all possible hypothesis). For this reason the inference theory for *johns-car* holds a second method based on analogy to be used when the first method fails. The preference to use first *car-g&t-diagnosis* is based in the knowledge that is a stronger method than analogy. If the MMA did not have this knowledge, it would choose one of both methods randomly, and after solving several car diagnoses cases it could base its preference on their successes and failures. [Syntactic remark: new objects or slots are written in bold, (define (foo bar) <body>) creates an object (or a slot) of name *bar* with body <body> and type *foo*. Anonymous objects are designated by its relation to a named one, e.g. the

"metatheory of johns-car" is designated as (meta reify of johns-car)].

```
(define (inference-theory
  (meta reify of johns-car ))
  (contents car-diagnosis-analogy-method
    car-diagnosis-inheritance-method)
  (define (link-select select)
    (link (>> reify of stronger-than))))

(define (theory (theory reify of johns-car))
  (type car-g&t-diagnosis))

(define (basic-analogy-method
  car-diagnosis-analogy-method ))

(define (type-inheritance-method
  car-diagnosis-inheritance-method)
  (stronger-than car-diagnosis-analogy-method))
```

The car-g&t-diagnosis holds the knowledge to generate malfunction hypothesis from complaints and test those hypotheses checking the facts known of a specific car (like the status of battery and gas tank). It also holds some preferences to choose among competing hypothesis. Hypothesis selection is based on knowledge about the estimated frequency of malfunctions, interpreting the more-frequent-than relation as a preference relation to select the current hypothesis.

```
(define car-g&t-diagnosis
  (complaint )
  (define (conditional battery-low?)
    (condition
      (>> low-voltage equal battery-voltage))
    (result true))
  (define (conditional no-gas?)
    (condition (>> empty equal gas-level)))
  (define (generate-&-test diagnosis )
    (define (complaint-to-malfunction-map
      generate-hypotheses)
      (complaint (>> complaint)))
    (define (select-hypothesis select)
      (link (>> reify of more-frequent-than)))
    (define (test-malfunction test-hypothesis)
      (device (>>)) ; the car itself
      (malfunction (>> select)))) ; cur. hypoth.
```

The generate-&-test method is a strategic cliché that retrieves its choice set by means of a generate-hypothesis method. The precedent-select method selects among the choice set using a method that retrieves a precedent case and prefers the hypothesis in the choice set according to their result (*successful/untried/failed*) in that precedent. The adapt process executes the test-hypothesis task to elucidate whether the current hypotheses is an adequate solution.

```
(define (strategic-cliche generate-&-test)
  (generate-hypotheses )
  (test-hypothesis )
  (contents (>> generate-hypotheses))
  (define (precedent-select select)) ;select hyp.
  (adapt (>> test-hypothesis))) ;test curr. hyp.
```

Generate-&-test method is a generic method and the select subtask uses a fairly general method. More specialized generate-&-test methods can be written with more focused selection. In car-g&t-diagnosis we can see that select-hypothesis first uses some knowledge-intensive criterion (like expected frequency of malfunctions) and if some unique selection cannot be achieved then a second method, precedent-select, is used to choose the best hypothesis.

```
(define (sequential-cliche select-hypothesis )
  (link )
  (contents (define (link-preference)
    (link (>> link)))
    (define (precedent-select )))))
```

The generate-hypothesis task uses the complaint-to-malfunction-map method that maps each complaint in our theory to the known set possible malfunctions that may cause it. This form of domain knowledge is very direct, and there could be other methods that derive this mapping from a causal model.

```
(define (strategic-cliche
  complaint-to-malfunction-map)
  (complaint )
  (contents (>> plausible-hypotheses complaint)))

(define (complaint does-not-start-complaint )
  (plausible-hypotheses
    low-battery-malfunction
    no-gas-malfunction))
```

Each car malfunction defines the test that can be used to verify it effectively occurs in a device, a repair recommendation and different relationships known to hold among car malfunctions (like their expected frequency, in the example below).

```
(define (car-malfunction low-battery-malfunction)
  (test
    (>> reify of (>> low-voltage equal battery-voltage)))
  (repair recharge-car-battery)
  (more-probable-than starter-malfunction))

(define (conditional-method test-malfunction)
  (device )
  (malfunction )
  (condition (>> (>> test malfunction) device)
    (result (>> malfunction)))
```

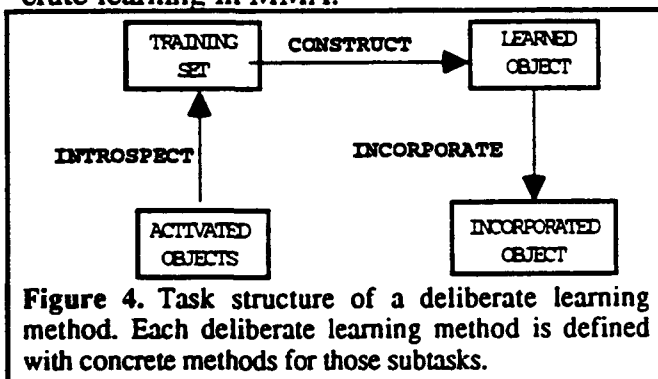
Now let us suppose that our theory about car diagnosis is incomplete (e.g. not all mappings

from complaints to malfunctions are complete), and that this is the case with *johns-car*. The failure of *car-g&t-diagnosis* causes the failure of the first inference method in metatheory of *johns-car*, and then the less preferred precedent-based method is selected. Retrieval of similar cases of car diagnosis can now retrieve casuistry information given while describing particular problem solving episodes. The *peters-car* precedent case asserts a simple causal explanation between the complaint *does-not-start* and the diagnosis known for that problem. Since the explanation of this case is in form of a method for diagnosis, *car-diagnosis-analogy-method* retrieves and applies this method to *johns-car* to see if that explanation also holds there.

```
(define peters-car
  (complaint does-not-start)
  (battery-voltage low-voltage)
  (define (causal-explanation diagnosis )
    (cause (> low-voltage equal battery-voltage))
    (effect low-battery-malfunction))
  (repair (>> repair malfunction diagnosis)))
```

5 Deliberate learning

The memorization of the episodes in problem solving constitutes the spontaneous and ubiquitous learning method used by MMA and required by MMA so as to function. We are currently experimenting with forms of deliberate learning integrated in MMA. *Deliberate learning* is made of learning methods (implemented as regular NOOS methods) that have to be explicitly called to be executed (typically, after finishing a task). The main purpose of Plaza and Arcos (1993) is to show the integration of memory and analogy through reflection, so in this section we are just going to sketch some of the essential features of deliberate learning in MMA.



Deliberate learning methods have in common a characteristic task decomposition shown in Fig. 4. The different deliberate learning methods are defined as an Introspect/Construct/Incorporate pattern. A specific method is obtained filling the task decomposition with specific methods for each subtask. Different learning methods are implemented with particular methods for (A) selecting a training set (introspect) from objects activated while solving a task in NOOS, (B) for constructing from them a new object, and (C) for incorporate that object to the rest of objects in the memory in an appropriate way.

6 Related work and Discussion

Our work on architectures is related to cognitive architectures like SOAR (Newell 90), THEO (Mitchell 91), and PRODIGY (Carbonell 91). At first sight, MMA language resembles THEO since NOOS is a frame language with caching, TMS, and "available methods" for slots. However, THEO does not provide a clear metaobject definition, does not reason about preferences over methods, and does not incorporate analogical reasoning or explicit inference methods. At a deeper level MMA resembles Soar in that MMA is a uniform, impasse-driven architecture with a built-in learning method. The differences are that spontaneous learning here is episode memorization and that our "learning as metalevel inference" hypothesis shapes another approach to inference and learning by the use of reification, self-models and the explicit representation of inference methods. The introspective use of meta-explanations in Meta-AQUA (Ram et al 92) is also related to MMA approach that exploits the reflective approach to learning. Meta-AQUA is not impasse-driven but proposes a mapping between classes of situations and learning methods that can improve the system. Meta-Router (Stroulia 92) combines planning and case-based reasoning in a task-decomposition framework and defines a typology of errors and methods for repair. Our current NOOS language is to be considered a descendant of languages RLL-1 (Greiner and Lenat 80) and KRS (van Marcke 87).

Related work on knowledge-level modelling of AI systems includes the Commet (or components of expertise) framework (Steels 90),

and the KADS methodology (Akkermans 92). Our approach is closer to the COMMET in that the ontology of models, tasks and methods proposed by COMMET is related to MMA's ontology of theories, methods and tasks. However, NOOS considers two layers: base-level domain theories and methods, and meta-level inference theories and methods, while the Commet approach is not reflective and only is concerned with the domain layer. The KADS methodology is much more different but they have used a reflective framework to describe the KADS four-layer architecture. However, neither Commet nor KADS have been used to perform learning tasks, and in fact MMA is the first attempt to apply knowledge level analysis to learning tasks and to develop a computational architecture that embodies that approach.

We have developed an architecture based on spontaneous learning by memorization of episodes and inference based on transfer from precedents. We are currently extending it to integrate other learning methods. Representation of learning methods and inference methods is uniform and based on conceptual frameworks previously used in the analysis and design expert systems and for knowledge acquisition. This "elements of inference" model allows reflection about inference, instead of previously used procedural or logic reflection. Reflection principles have been used to provide a computational mechanism to integrate learning and inference in a uniform representation. Reflection is also being used to model in a principled way the relationship between learning components and the architecture as a whole.

Acknowledgements

The research reported on this paper has been developed at the IIIA inside the Massive Memory Project (CICYT grant 801/90).

References

Akkermans, H., van Harmelen, F., Schreiber, G., Wielinga, B., "A formalisation of knowledge-level model for knowledge acquisition". *Int. J. Intell. Sys.* 8(2): 169-208, 1993.

Carbonell, J G, Knoblock, C A, Minton, S, "Prodigy: An integrated architecture for planning and learning."

In K Van Kehn (Eds.), *Architectures for Intelligence*. Lawrence Erlbaum Ass., Hillsdale, NJ, 1991.

Feferman, S, "Transfinite recursive progressions of axiomatic theories". *Journal of Symbolic Logic*, 27:259-316, 1962.

Greiner, R., Lenat, D. "RLL-1: A Representation Language Language", HPP-80-9 Comp. Science dept., Stanford University, 1980.

Michalski, R S, "Inferential learning theory as a basis for multistrategy task-adaptive learning". In R Michalski and G Tecuci (Eds.) *Proc. Int. Work. on Multistrategy Learning*, p. 3-18, 1991.

S Minton, J Carbonell, C Knoblock, D Kuokka, O Etzioni, Y Gil, "Explanation-based learning: A problem solving perspective", *Artificial Intelligence*, 40, 63-118, 1989.

Mitchell, T.M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., Schlimmer, J. C., "Theo: a framework for self-improving systems". In K Van Lenhn (Ed.) *Architectures for Intelligence*. Laurence Erlbaum, 1991.

Newell, A, *Unified Theories of Cognition*. Cambridge: Harvard Univ. Press, 1990.

Plaza, E, "Reflection for analogy". *Proc. IMSA'92 Workshop on Reflection and Metalevel Architectures*, Tokyo, November 1992, p. 166-171, 1992.

Plaza, E, *Constitutional Axioms and Rules for NOOS*. IIIA Research Report 92/15, 1992b.

Plaza, E, Arcos, J L, *Reflection, Memory, and Learning*, IIIA RR 93/2, 1993.

Ram, A, Cox, M T, Narayanan, S, "An architecture for integrated introspective learning". *Proc. ML'92 Workshop on Computational Architectures for Machine Learning and Knowledge Acquisition*, 1992.

Smith, B.C. "Reflection and semantics in a procedural language". In Brachman, R. J., and Levesque, H. J. (Eds.) *Readings in Knowledge Representation*. Morgan Kauffman, pp. 31-40, 1985.

Steels, L, "The Components of Expertise", *AI Magazine*, Summer 1990.

Stroulia, E and Goel, A K, An architecture for incremental self-adaptation. *Proc. ML-92 Workshop on Computational Architectures for Machine Learning and Knowledge Acquisition*. 7/4/92: Aberdeen, Scotland, 1992.

van Marcke, K, "KRS: An object-oriented representation language". *Revue d'Intel. Artif.* 1(4), 43-68, 1987.

Weyhrauch, R W (1989), "Prolegomena to a theory of mechinazed formal reasoning". *ARTINT*, 13(2).

Wielinga, B, Schreiber, A, Breuker, J, "KADS: A modelling approach to knowledge engineering". *Knowledge Acquisition* 4(1), 1992.

Learning Scope, Task Analysis, and Sharable Components

Erik M Altmann
School of Computer Science
Carnegie Mellon University
altmann@cs.cmu.edu

Abstract

Problems facing multistrategy learning include managing the complexity of multiple interacting learning components and managing the cost of implementing them. High-level approaches to these two problems are presented. Analysis of the task domain can generate constraints on how performance and learning components interact. This analysis can be verified with respect to human performance, providing a standard against which to measure the scope of learning in the system. The marginal cost of adding new varieties of learning to a system can be reduced by using sharable learning components, like those embedded in general problem-solving architectures. These approaches require a meaningful definition of what constitutes learning variety, in order that systems can be compared and that system complexity and implementation cost can be evaluated with respect to the scope of learning that the system achieves. Two constituents of learning scope, density and diversity, are proposed.

Keywords: Multistrategy learning, human problem solving, task analysis, architectures

1. Introduction

Multistrategy learning faces some hard problems. One is that of getting multiple

kinds of learning in one system to coordinate and communicate effectively. Various specific solutions have been investigated, including hard-wiring control of the components (Hammond, 1989; Pazzani, 1990) and opening control to the user (Morik, 1991). A more general methodological problem is to develop constraints on system functionality that help shrink the design space of possible solutions. Emulating humans' ability to perform and learn flexibly is one approach to developing such constraints.

A second problem in multistrategy learning is the cost of implementing multiple kinds of learning in a given system. Cost increases with the variety of learning forms, reflecting the need to implement additional unique learning algorithms. Lowering the marginal cost of adding new kinds of learning to a system will be an enabling factor in increasing the learning scope of systems. Architectural learning components will play a role in reducing this marginal cost.

Bearing on both these problems is that the notion of a variety of kinds of learning, or **learning scope**, should be a well-defined, measurable quantity. Some means of evaluation is necessary. The learning scope of a system consists of at least two measurable parameters, **density** and **diversity**.

Section 2 defines density and diversity. Section 3 proposes that a useful taxonomy for measuring diversity derives from an analysis of the domain at hand. Section 4 gives a sample analysis that generates such a taxonomy. Section 6 discusses the role of sharable learning components in reducing implementation cost. Section 7 draws together implications of the discussion.

2. A Definition of Learning Scope

Learning scope has at least two constituents. First, scope implies learning with respect to as many of a system's aspects of performance as possible. This quality will be called **density**. A measure for density is the **proportion of components in a system that are learning targets**, where a **learning target** is a system component (process or data structure) whose behavior changes over time. Density reflects the extent to which learning pervades the system. One-hundred percent density would solve the "wandering bottleneck" problem (Mitchell, 1983), in which a performance bottleneck arises wherever there is a component that is not a target.

Learning density by itself fails to capture at least one intuition about what constitutes scope: a system with many *similar* learning targets may be learning-dense, but in a way that is degenerate in terms of variety.

Therefore, a second constituent of scope must measure the **diversity** in a system's learning behavior. Measuring diversity requires an objective standard, which should be external to the system so that it can be used to compare systems. Such a standard would have the character of a taxonomy that categorizes kinds of learning. Relative to such a taxonomy, **diversity is the extent to which a learning system covers the taxonomy's categories**. Diversity complements density, providing a means of evaluating the significance of having many learning targets.

3. Domain-Specific Taxonomies

A taxonomy for measuring learning diversity can be derived from an analysis of the system's domain. This measure is justified by the following knowledge-level perspective. Learning components generate knowledge for a target. A target is a body of performance knowledge that grows with the output of some learning component. Each target represents a different body of knowledge, so targets provide a dimension of knowledge-level variability. This dimension constitutes a measure of learning diversity. Moreover, it explicitly reflects the kinds of knowledge processed through learning, complementing domain-independent taxonomies like those of Langley (1987) and Carbonell et al (1983).

A domain-specific taxonomy needs to be constructed anew for each domain, using task analysis. One methodology is as follows. The task analyst first finds features that the domain *affords* (offers) for problem solving methods to manipulate. Domains differ in what features they afford. For example, simple symbolic-logic domains afford the detection of differences between current and desired states, making means-ends analysis (MEA) feasible; in chess, on the other hand, the desired state is too vague to afford difference detection, so MEA is not feasible (Newell and Simon, 1972).

With affordances in hand, the analyst then finds methods that use them (such as MEA). Methods, applied in a particular domain, are made up of components of performance knowledge. These constituents are learning targets. The collection of such targets, from all the methods that apply to the domain, form the domain-specific taxonomy.

Learning diversity relative to this taxonomy depends on performance diversity; the more varied the performance components, the more varied the set of learning targets. This may seem to just move the difficulty from the

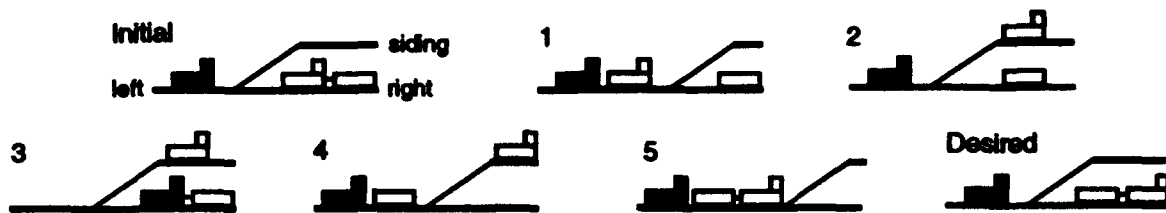


Figure 1: Simple task from the switchyard domain

measurement of learning diversity to the measurement of performance diversity. But task analysis, while it cannot directly specify how a system will *learn* when operating in a particular task environment, can specify how a system must *perform* in that environment (Newell and Simon, 1972; Wilde and Lewis, 1991). The notion that task analysis can define requirements for system performance underlies Anderson's rational-analysis theory of cognition (Anderson, 1990). Similarly, diversity in performance as a response to a complex environment is a subtext of Minsky's Society of Mind (Minsky, 1986). Thus task analysis has a history of being used to analyze performance. With respect to learning, analyzing performance is an indirect but operational way to measure learning diversity.

Moreover, task analysis is itself amenable to verification against an external standard. Human performance can be monitored experimentally through protocols (Ericsson and Simon, 1984), and analyzed to model the underlying computational behavior (Newell and Simon, 1972). Performance specifications derived from task analysis can therefore be calibrated against models of human performance.

This calibration is relative to the human scale. While this scale is not the limit of performance and learning, it provides an

external standard of proficiency that computer systems have not yet reached.

4. A Sample Domain-Specific Taxonomy

This section gives an example of a domain-specific taxonomy by sketching several problem-solving methods for a domain, in enough detail to show high-level components and how the method is useful. These methods reveal a set of learning targets for the domain. In total there are five methods and ten targets.

Tasks in this domain manipulate trains in a railway switchyard. The sample task analysis shows how affordances in the domain (such as difference detection) are manipulated by methods, and how even a simple, knowledge-lean domain can yield a variety of targets.

Figure 1 shows a simple task in which the two unshaded cars swap positions. The solution path is included; a step consists of a train moving from one position (left, right, or siding) to another.

Figure 2 shows a puzzle task (Delft and Botermans, 1978), in which two trains need to pass each other when the siding is too small for a full train. The optimal solution has sixteen moves.

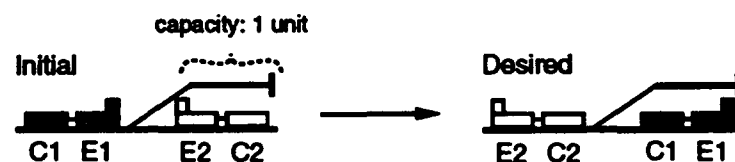


Figure 2: Puzzle task from the switchyard domain

Figure 3: M_1 must be undoneFigure 4: M_1 is necessary

1. Heuristic search (1 target). Heuristics are knowledge about what steps are or are not usually useful. One powerful heuristic for the puzzle task prefers moving trains with one engine to moving trains with two. The pool of heuristic knowledge is a learning target. A learning component for this target could compile heuristics from past experience, or specialize them from domain-independent heuristics in the manner of Eurisko (Lenat, 1983).

2. Case-based search control (3 targets). Case-based reasoning (CBR) makes use of a memory of past problem solving episodes (cases). One kind of useful case comprises a sequence of moves, the task to which the sequence was applied, and the result, whether positive or negative (Veloso and Carbonell, 1991). Figure 3 shows two successive moves from the initial state of the puzzle task, applying the heuristic from the previous paragraph to choose the second move (M_2). The heuristic in this case leads to a state loop. To prevent this loop from recurring, either immediately or later in this or another task, M_1 and its outcome under this heuristic can be learned, for recall under similar circumstances.

Case memory itself is a target, because the more cases learned the better (if they are stored and retrieved properly). When a case is stored, a component must decide how the case will be indexed. This component is also a target, because good indexing depends on knowledge about what features of a case are relevant, and this knowledge can be learned. And at retrieval time, another component must search for relevant cases. But cases that seem relevant may be misleading, because indexing cannot be perfect. For example, in Figure 4 the move M_1 , which was a dead end in Figure 3, is necessary to allow the engine on the siding to move past the left-hand train. Thus the retrieval component is also a target, because, as it learns more about the domain, it can retrieve cases more accurately.

3. Means-ends analysis (3 targets). Means-ends analysis (MEA) requires that the current state be compared with a desired state and differences detected. In Figure 5, the unshaded engine begins to the right of the flatcar but ends up not to the right. Reducing this difference entails getting the two units past each other, but the steps needed lead directly to the desired state. MEA is particularly powerful in this case.

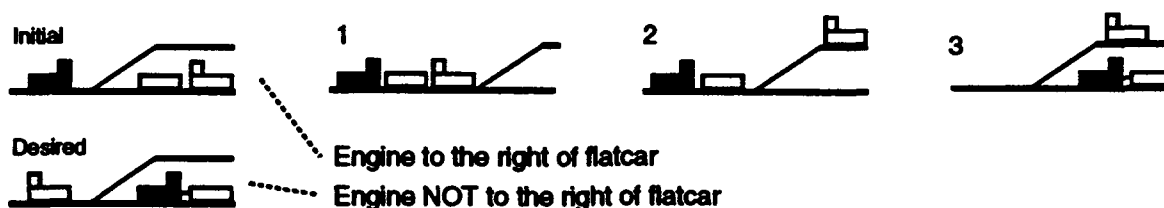


Figure 5: Applying means-ends analysis

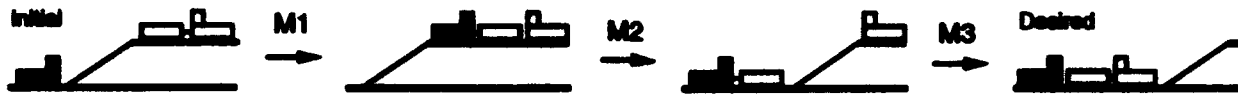


Figure 6: A correct but inefficient solution

Components for MEA include difference detection, difference ranking, and the connections between operators and the differences they reduce. All three are targets (see also Rich 1983, p 367, for a discussion of learning in MEA).

4. Solution refinement (1 target). The switchyard domain admits solutions that are correct but inefficient. In Figure 6 the task is solved in three steps, but can be solved in one by moving the unshaded train down from the siding directly. Humans are adept at recognizing and avoiding simple inefficient sequences like this one. In general this requires combinatoric search, making solution refinement a good target: the problem solver can get better at avoiding inefficient sequences.

5. Strategy selection (2 targets). Given multiple methods, the problem solver must be able to make tactical decisions about when to choose a particular method and when to compose methods (an example of composition would be the application of MEA during solution refinement). It must also evaluate whether a method is making progress and abandon it if not. The problem solver can get better at selecting methods and smarter about abandoning them.

5. The Uses of Task Analysis

Task analysis as demonstrated in the previous subsection serves several purposes, summarized here. First, it provides an external standard against which to measure learning diversity; a system learns diversely to the extent that its kinds of learning cover the targets identified through task analysis. This external measure of scope complements density, which provides a measure internal to

the system.

Second, it provides a specification of performance methods for that domain. The more detailed the task analysis, the more detailed the specification.

Third, and related to the previous point, it provides a specification for how to arrange communication and control between methods. In the switchyard example, strategy selection (method 5) received only perfunctory treatment, but in principle task analysis could yield considerable guidance for how to organize the other methods to work together productively. This makes task analysis an important source of leverage on the hard problems of integrating multiple methods.

Fourth, task analysis can be verified for completeness relative to human performance. This verification serves to calibrate a domain-specific taxonomy to the human scale.

6. Sharable Learning Components

In any kind of system-building, cost can be reduced by sharing (reusing) modules. For learning systems, cost can be reduced by sharing learning components. Sharable learning components today are typically found embedded in general problem-solving architectures. Examples include chunking in Soar (Laird et al, 1986), the Labyrinth conceptual-clustering module in Icarus (Langley et al, 1991), and the various learning modules in Prodigy (Carbonell et al, 1991).

Two factors affect the promise of a particular architecture to support broad learning scope. First is the extent to which the architecture *reduces the overhead* of expanding the learning scope of a system. Ideally, adding a

new performance method would be routine. So would connecting it to a sharable learning component to create one or more learning targets. In fact, however, routine addition of new methods and routine integration with learning components has not been achieved. For example, correct integration of chunking into Soar performance systems is highly sensitive to the representation of performance knowledge (Laird et al, 1986). But this sensitivity may be endemic to learning systems (Flann and Dietterich, 1989); if so, the relative promise of sharable components remains.

The second factor affecting an architecture's potential for supporting scope is the extent to which its sharable learning components are *general*. The more a component can learn about, the more targets are possible. Generality depends on how the component is integrated with other components in the architecture.

Chunking, for example, is integrated with Soar's control strategy, *universal subgoal*ing (Laird, 1984). In universal subgoaling, a gap or complexity in performance knowledge prompts the architecture to set a subgoal. When problem-solving achieves the subgoal, chunking captures the element of knowledge, whether compiled from knowledge elsewhere in the system or brand new (Newell, 1990). Thus, although chunking has only one architectural target (production memory), it has in principle an unbounded number of performance-knowledge targets. So far, it has been applied (in different systems) to instruction taking (Huffman, 1991), induction and knowledge-level learning (Rosenbloom and Aasman, 1990; Miller and Laird, 1991), and integration of multiple sources of knowledge in natural language understanding (Lehman et al, 1991), as well as speedup learning (Laird et al, 1986; Tambe, 1991). (A summary of references for varieties of learning in Soar appears in the introduction to

Rosenbloom et al, 1993).

The accumulated evidence for chunking suggests that it is general enough to support unbounded increases in scope, but that possibility has not been proven. The promise of mechanisms like chunking has yet to be tested in an effort aimed directly at achieving great scope in a single system.

7. Implications for Multistrategy Learning

Two observations and an ensuing hypothesis can be drawn from this discussion of scope measures and sharable learning components.

The first observation is that solutions to problems like integration and control of multiple learning components depend as much on the domain as they do on generic components. This is borne out in the task analysis, in the need for a strategy selection method (page 5). Here the perfunctory treatment of this method hides many of the problems of getting the other methods to perform and learn together, but more detailed task analysis would help define these problems and guide solutions. General solutions to these hard problems may emerge only through pursuit of broad learning scope on a variety of domains.

The second observation is that sharable learning components are potentially better at enabling great scope than special-purpose, hardwired implementations of learning algorithms. Relative to a domain-specific measure of learning scope, the variety of *generic* learning algorithms in a system is a secondary issue; as few as one might do to achieve great scope relative to the domain at hand.

These two observations lead to the following hypothesis: A direct and comparatively low-overhead way to face hard problems in multistrategy learning, such as control and

integration of learning components, is to pursue broad learning scope for specific domains within architectures, guided by comprehensive task analysis.

8. Summary

Broad learning scope involves modifying a variety of bodies of performance knowledge. Two qualities of scope are density and diversity. Density measures the degree to which learning pervades a system. Diversity measures the degree to which learning covers the kinds of learning afforded by the domain.

A domain-specific taxonomy for measuring learning diversity is a product of task analysis, which also helps specify performance methods, including those responsible for control flow and communication. Task analysis can be checked for completeness against human behavior, and serves to calibrate a domain-specific taxonomy of learning forms to human-scale performance.

Building systems with broad scope is expensive because of the high marginal cost of building new system components to modify new targets. Sharable learning components, as embedded in learning architectures, hold the promise of reducing this overhead.

Sharable learning components, providing cost advantage, and task analysis, providing method specifications, guidance for integration and control, and means of measurement, both need to be exploited in the building of systems with broad learning scope.

Acknowledgements

Allen Newell was a collaborator in this research. Thanks to Bob Doorenbos, Andrew Howes, Rick Lewis, Gary Pelton, Paul Rosenbloom, and the MSL-93 reviewers for their comments.

This research was sponsored in part by the Avionics Laboratory, Wright Research and

Development Center, Aeronautical Systems Division (AFSC), U S Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The research was also supported in part by the Natural Sciences and Engineering Research Council of Canada. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, the U S Government, or the Government of Canada.

References

- Anderson, J R. *The Adaptive Character of Thought*. Hillsdale, NJ: Erlbaum. 1990.
- Carbonell, J G, Michalski, R C, and Mitchell, T M. An overview of machine learning. In R C Michalski, J G Carbonell, and T M Mitchell (Eds), *Machine Learning: An artificial intelligence approach*. Palo Alto: Tioga. 1983.
- Carbonell, J G, Etzioni, O, Gil, Y, Joseph, R, Knoblock, C, Minton, S, and Veloso, M M. PRODIGY: An integrated architecture for planning and learning. *SIGART Bulletin*, Vol 2, No 4, pp 51-55. 1991.
- Delft, P van and Botermans, J. *Creative Puzzles of the World*. New York: Abrams. 1978.
- Ericsson, K A and Simon, H A. *Protocol Analysis: Verbal reports as data*. Cambridge: MIT Press. 1984.
- Flann, N S and Dietterich, T G. A study of explanation-based methods for inductive learning. *Machine Learning*, Vol 4, No 2, pp 187-226. 1989.
- Hammond, K J. *Case-Based Planning*. New York: Academic Press. 1989.
- Huffman, S B. Integrating perception, language and action in Soar: A preliminary report. Thesis proposal, The University of Michigan Artificial Intelligence Laboratory.

1991.

Laird, J E. *Universal subgoalting*. Doctoral dissertation, Carnegie Mellon University School of Computer Science. 1984.

Laird, J E, Rosenbloom, P S, and Newell, A. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, Vol 1, No 1, pp 11-46. 1986.

Langley, P. Learning, Machine. *Encyclopedia of Artificial Intelligence*, Vol 2, pp 463-487. 1987.

Langley, P, McKusick, K B, Allen, J A, Iba, W F, and Thompson, K. A design for the ICARUS architecture. *SIGART Bulletin*, Vol 2, No 4, pp 104-109. 1991.

Lehman, J F, Lewis, R L, and Newell, A. *Natural Language Comprehension in Soar: Spring 1991* (Tech Rep CMU-CS-91-117). Carnegie Mellon University School of Computer Science. 1991.

Lenat, D G. The role of heuristics in learning by discovery: Three case studies. In R C Michalski, J G Carbonell, and T M Mitchell (Eds), *Machine Learning: An artificial intelligence approach*. Palo Alto: Tioga. 1983.

Miller, C S and Laird, J E. A constraint-motivated model of lexical acquisition. *Proceedings of the Eighth International Workshop on Machine Learning* (pp 95-99). San Mateo, CA: Morgan Kaufman. 1991.

Minsky, M L. *The Society of Mind*. New York: Simon and Schuster. 1986.

Mitchell, T M. Learning and problem solving. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp 1139-1151). Los Altos, CA: William Kaufman. 1983.

Morik, K. Balanced cooperative modeling. *Proceedings of the First International*

Workshop on Multistrategy Learning (pp 65-80). Fairfax, VA: George Mason University. 1991.

Newell, A. *Unified Theories of Cognition*. Cambridge: Harvard University Press. 1990.

Newell, A and Simon, H A. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall. 1972.

Pazzani, M J. *Creating a Memory of Causal Relationships: An integration of empirical and explanation-based learning methods*. Hillsdale, NJ: Erlbaum. 1990.

Rich, E. *Artificial Intelligence*. New York: McGraw-Hill. 1983.

Rosenbloom, P S and Aasman J. Knowledge level and inductive uses of chunking (EBL). *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp 821-827). Cambridge: MIT Press. 1990.

Rosenbloom, P S, Laird, J E, and Newell, A (Eds). *The Soar Papers: Research on integrated intelligence (Vol 1)*. Cambridge: MIT Press. 1993. In Press.

Tambe, M. *Eliminating Combinatorics from Production Match*. Doctoral dissertation, Carnegie Mellon University School of Computer Science. Tech Rep CMU-CS-91-150. 1991.

Veloso, M M and Carbonell, J G. Automating case generation, storage, and retrieval in PRODIGY. *Proceedings of the First International Workshop on Multistrategy Learning* (pp 363-377). Fairfax, VA: George Mason University. 1991.

Wilde, N P and Lewis, C H. *Inventing Information Representations Through Task Analysis* (Tech Rep CU-CS-549-91). University of Colorado at Boulder Department of Computer Science. 1991.

INTELOG: A Framework for Multistrategy Learning

Wayne T. Jenkins
 Royal Melbourne Institute of Technology
 GPO Box 2476V
 Melbourne, Australia, 3001
 wtj@goanna.cs.rmit.oz.au

Abstract

This paper introduces the concept of multisource learning. It demonstrates how a multisource learner can be used as a framework for multistrategy learning. A variation of the PAC model of learning correctness called PAC-Error identification is defined and shown to be suitable for multisource learning. An algorithm for PAC-Error identification is developed for a knowledge base of definite clauses. It is shown that PAC-Error identification is effective in integrating definite clauses that have been contributed from several sources. This framework can be used as the basis of a multistrategy learner which can produce hypotheses with small error.

Keywords: Multistrategy Learning
 Multisource Learning
 PAC Learning

1 Introduction

A multistrategy learner combines learning strategies to produce effective hypotheses. If each of the learning strategies was implemented as an independent entity that contributed knowledge to a knowledge base then an algorithm would be required to integrate this knowledge to produce an effective hypothesis. A multisource learner implements such an algorithm. Effectiveness is measured as the ability of the resultant hypothesis to sat-

isfactorily complete performance tasks.

The original idea behind multisource learning was to permit knowledge sources, which were themselves independent learning elements, to contribute their knowledge to a knowledge base. It was envisaged that the knowledge base would contain multiple target concepts and the learning elements could be used to build different parts of the knowledge base. Each learning element could also implement a different strategy. For example, an explanation based learner could be used to improve efficiency, an abductive learning element would contribute new, plausible clauses and an inductive learning element could construct new clauses from examples.

For our purposes we assume that the knowledge base is a set of definite clauses.

A multisource learner must select those clauses from the knowledge base which result in satisfactory performance across the population of tasks. Collectively the selected clauses are a hypothesis.

A major issue in extracting a hypothesis is in determining a model of correctness. A logic programming definition could be that all logical consequences of a hypothesis h are in the intended interpretation and all elements of the intended interpretation are logical conse-

quences of h . There are two problems with this definition:

1. The hypothesis can often be more general than what is required. It requires only those clauses relevant to the set of tasks to be performed.
2. Exact identification of the hypothesis can be too strong as the criterion for correctness. A weaker variant may be more practical.

This paper outlines a model for multisource learning and presents an algorithm for what can be termed PAC-Error identification. This algorithm is based on a PAC correctness model.

2 A Model of Learning

A multisource learner is shown in Figure 1.

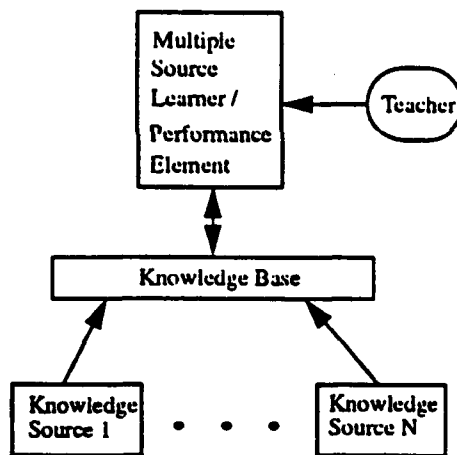


Figure 1

Its knowledge base contains a set of definite clauses. Each knowledge source independently contributes definite clauses to the knowledge base. The multisource learner accepts tasks in the form of classified goal clauses from the teacher. The aim of the multisource learner is to extract a set of definite clauses from the knowledge base that satisfactorily (within the constraints of the PAC model) solve the tasks given by the teacher.

In the case of multistrategy learning the knowledge sources implement various learning strategies. The details of these knowl-

edge sources (in this case learning elements) are beyond the scope of this paper other than to show the contributions that each learning element makes to the knowledge base.

The multisource model is a supervised learning model. Typically, in supervised models, examples are vectors of attribute-value pairs and the classification indicates whether the example is a member of the target concept or not. The examples given by the teacher in multisource learning are goal clauses where a simple boolean flag is not sufficient for classification. Instead the teacher must provide the set of goal clause instances which are members of the target concept.

Example 1: Suppose that a knowledge base is to contain facts about relationships between family members. A goal clause could be $\leftarrow \text{father}(\text{fred}, C)$. The instances which are a member of the target concept could be $\{\text{father}(\text{fred}, \text{ian}), \text{father}(\text{fred}, \text{jane})\}$.

3 Correctness of Learning

Traditionally, the ideal for the correctness of a hypothesis has been that it determine all positive members of a target concept to be positive and negative members to be negative. The sample complexity for this task can be large. No matter how many examples are used to derive a hypothesis an example can be presented which invalidates the hypothesis.

Probably approximately correct (PAC) learning (Valiant, 1984) permits a probabilistic characterization of a hypothesis. After n examples the probability of being presented with an example that is inconsistent with the derived hypothesis is less than an error threshold ϵ with a probability $1 - \delta$. The literature contains many results for learning classes of hypotheses (Kearns, 1989).

In multisource learning, as described in this paper, the hypothesis class H is the power set

of the set of definite clauses in the knowledge base. A PAC learning algorithm would require that a hypothesis be identified from this set that has error less than ϵ with high confidence. Clearly, for multisource learning, it is not always possible to find such a hypothesis in H because H is dependent on the clauses available in the knowledge base. For multisource learning the hypothesis should have the minimal error in the set of hypotheses. The multisource learner, described in this paper, performs what can be termed PAC-Error identification where it returns a hypothesis h that with confidence $1 - \delta$ has an error probability in the interval $[\epsilon - d, \epsilon + d]$. The constant d determines the closeness with which the error is to be approximated. Ideally the hypothesis h has the smallest ϵ in H .

4 Probability Regions

A goal clause, g , permits several instances¹. Each goal clause has a set of instances which are members of the target concept. Denote this set by T_g . A hypothesis, h , determines the set of goal instances, h_g that logically follow from it.

Each goal instance has a relative probability defined to be the probability of its parent goal divided by the number of instances possible for the parent goal. The total probability of a goal instance is the sum of relative probabilities of all its parent goal clauses.

Figure 2 shows the universe of instances of those goal clauses available to the teacher and the relationship between T_g and h_g .

Let T_u be the set of all goal instances from U that are members of the target concept and h_u be the set of all instances from U that logi-

cally follow from h .

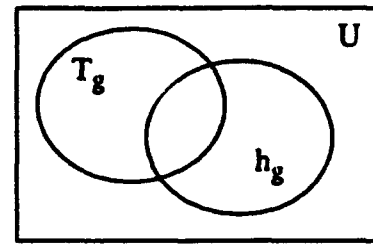


Figure 2

Four probability regions are defined:

False positive region (FP): Denotes the set of instances that are in h_u but not in T_u . Let FP-probability be the sum of the probabilities of the instances in this region.

True negative region (TN): Denotes the set of instances that are not in h_u and not in T_u . Let TN-probability be the sum of the probabilities in this region.

True positive region (TP): Denotes the set of instances that are in h_u and T_u . Let TP-probability be the sum of the probabilities in this region.

False negative region (FN): Denotes the set of instances that are not in h_u but are in T_u . Let FN-probability be the sum of probabilities in this region.

The error probability of a hypotheses h is the sum of the FP-probability and the FN-probability.

Figure 3 shows the probability regions across the universe of goal instances.

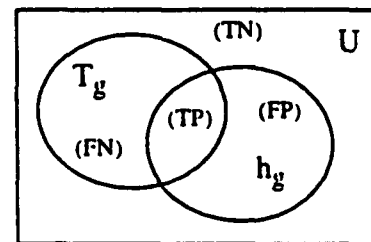


Figure 3

There are two observations about these probability regions that aid in the development of a learning algorithm:

1. The deletion of a clause can result in the

1. A ground instance is derived by substituting constants for all variables in the goal clause. For this paper assume all instances are ground.

transfer of TP-probability to FN-probability and FP-probability to TP-probability.

2. The addition of a clause can result in the transfer of FN-probability to TP-probability and TN-probability to FP-probability.

5 Determining Confidence in the Error

Consider a population of examples which are either classified as positive or negative with respect to some target concept. What is the probability that a randomly chosen example will be incorrectly classified by a hypothesis h . This is the error rate for the hypothesis h .

The error rate for a hypothesis h can be approximated by sampling. Suppose that n examples are chosen at random from the population. Let X_i be 0 if example i is correctly classified and 1 otherwise. Each X_i is a random variable and $\langle X_1, X_2, \dots, X_n \rangle$ is a sample of the population. The sample mean is \bar{X} . The X_i are independent and identically distributed.

The expected probability of error is the sum of the probabilities of incorrectly classified examples across the population. This mean can be estimated and a confidence limit for it can be determined through sampling.

Assume that $n \geq 30$ and the estimate of standard deviation s has inconsequential error with respect to σ . The distribution of the sample mean \bar{X} is approximately normal. Two theorems are useful in this case (Walpole et. al., 1978).

Theorem 1: If \bar{x} is used as an estimate of μ , we can be $(1 - \delta)100\%$ confident that error will be less than

$$z_{\delta/2} \frac{\sigma}{\sqrt{n}}$$

The value $z_{\delta/2}$ is the value of the standardized normal variable below which there is an area of $\delta/2$.

Theorem 2: if \bar{x} is used as an estimate of μ we can be $(1 - \delta)100\%$ confident that the error will be less than a specified amount d when the sample size is:

$$n = \left(\frac{z_{\delta/2} \sigma}{d} \right)^2$$

Example 2: Suppose that a sample of 40 examples was taken from the population of examples. Ten of these examples are incorrect. The sample standard deviation s is 0.1923. Assuming that s accurately estimates σ then the number of examples required such that the difference between the sample mean and the true mean is less than 0.01 with confidence 0.95 is:

$$n = \left(\frac{z_{\delta/2} 0.1923}{0.01} \right)^2 = 1421$$

The sample mean is the error rate.

These confidence calculations apply equally to the case where each X_i has an error value in the range $[0, 1]$. This is the case for goal clauses. Each goal instance of a goal clause g has a value which is $1/(\# \text{ instances of } g)$. This value assumes that each instance of a goal clause is equally important. The error value for the goal clause is the sum of values of those goal instances in $h_g \Delta T_g$ (the symmetric difference). The expected value for this error value gives the error rate across goal clauses. The error rate can be interpreted as the probability that an arbitrary goal clause g will have instances in T_g not in h_g or instances in h_g not in T_g .

6 PAC-Error Identification

The algorithm presented here performs PAC-Error identification of a hypothesis $h \subseteq C$ where C is the set of clauses available in the knowledge base.

Roughly, the algorithm calculates the TP, FP, FN and TN probabilities with $h = C$. It main-

tains sufficient information to determine all probabilities of any $h' \in H$. A hill-climbing search is applied to obtain a local optimum of ϵ and the final hypothesis h_k is returned as the result. The required information includes the TP and FP probabilities of each clause and the set of refutation sets (a refutation set is the set of clauses involved in a refutation of a goal instance). Each iteration of the algorithm deletes a clause until no further deletions improve the quality of the hypothesis. A sequence of hypotheses is derived:

$$\langle (h = h_1), h_2, \dots, h_k \rangle$$

Each element h_i in the sequence has less error and one less clause than its predecessor. Each element h_i has values for TP, FP, FN and TN. The resultant hypothesis h_k is returned with its error probability ϵ .

The algorithm has two major components: determining region probabilities with the desired confidence (ConfThresh) and identifying the clause set that has small error (PAC-Error Ident).

ConfThresh(δ, d)

1. Randomly select $m \geq 30$ goal clauses, calculate the error of each and find the sample standard deviation s . Calculate the number of samples n required to obtain the necessary confidence $1 - \delta$ for the interval size $2d$.
2. Initialise the region values² TP, FN, FP and TN to 0.
3. Let h be the set of clauses in the knowledge base.
4. Perform n times:
 - (a) Accept a classified goal clause from the teacher. T_g is given.
 - (b) Derive the set of goal instances h_g implied by h . Retain the refutation set R_{gi} of every goal instance i .
 - (c) Calculate the value of each goal instance g_i . Let this value be denoted by v_{gi} .
 - (d) For each goal instance, $g_i \in h_g$:
 - If $g_i \in T_g$ increment TP by v_{gi} . Also increment the TP of each clause in R_{gi} and the TP

of R_{gi} by v_{gi} .

- If $g_i \in T_g$ increment FP by v_{gi} . Also increment the FP of each clause in R_{gi} and the FP of R_{gi} by v_{gi} .

(e) For each $t_i \in T_g - h_g$ increment FN by v_{ti} .

5. Divide TP, FP, FN and TN by the sample count n .

6. Perform PAC-Error-Ident (TP, FP, FN, TN, h).

It must be noted that when maintaining probabilities for clauses and refutation sets only the probabilities TP and FP have any meaning.

PAC-Error-Ident(TP, FP, FN, TN, h)

1. Let $i = 1$.
2. Let $h_i = h$.
3. Let TP_k be the TP value of clause C_k .
4. Let FP_k be the FP value of clause C_k .
5. Let $Error_k$ be $FP_k - TP_k$ for clause C_k .
6. Let $C_j \in h_i$ be the clause with the highest error value $Error_j$ for all j .
7. Perform until no clause $C_j \in h_i$ has $Error_j \geq 0$.
 - (a) For each clause $C_r \in h_i$ where $r \neq j$ and C_j participates in a refutation with C_r :
 - Subtract the TP-probability of every common refutation from the TP-probability of clause C_r
 - Subtract the FP-probability of every common refutation from the FP-probability of clause C_r
 - (b) Let $TP = TP - TP_j$
 - (c) Let $FP = FP - FP_j$
 - (d) Let $FN = FN + TP_j$
 - (e) Let $TN = TN + FP_j$
 - (f) Let $i = i + 1$.
 - (g) Let $h_i = h_{i-1} - \{C_j\}$
 - (h) Let $C_j \in h_i$ be the clause with the highest error value $Error_j$ for all j .
8. Return h_i and $\epsilon = FP + FN$.

7 Experimental Work

7.1 A domain theory

Consider the family tree given in Figure 4. Assume that the multistrategy learner must learn the relationships that hold in this tree. It is assumed that a language is pre-defined for

2. Region values only become probabilities after division by the sample count, n .

definite clauses and goal clauses.

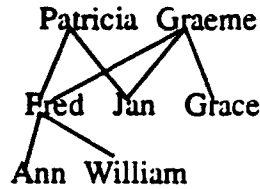


Figure 4

The knowledge base initially contains the following clauses:

```

sister(X,Y) ← parent(Z,X),parent(Z,Y), female(X)
sister(jan, fred) ←
parent(patricia, grace) ←
parent(fred, ann)
parent(fred, william) ←
parent(graeme, fred) ←
male(fred) ←

```

For illustration purposes assume that the population of goal clauses, their classifications and probability distribution are known and given in Table 1.

Goal Clause (g)	T _g	Prob.
father(fred, X)	father(fred, ann) father(fred, william)	0.2
father(X, jan)	father(graeme, jan)	0.3
mother(X, fred)	mother(patricia, fred)	0.3
sister(jan, X)	sister(jan, grace) sister(jan, fred)	0.2

Table 1: Goal Clause Probabilities

Here we use the probability distribution to analytically determine the sample size. Normally the probability distribution would not be known a priori. The ConfThresh algorithm would be needed to determine the sample size.

The region probabilities are the expected values of each region value across goal clauses.

Example 3: Consider the goal clause \leftarrow father(fred, X). There are seven possible solutions to this goal clause (all constants in the language i.e. all family tree members). The value of any one solution is 1/7.

The original domain theory does not entail any

instances of this goal clause but there should be two (ann and william). The FN-value is 2/7, the FP-value is 0, the TP-value is 0 and the TN-value is 5/7.

7.2 Knowledge sources/learning elements

The multisource learner becomes a multistrategy learner when it permits knowledge sources to become independent learning elements. The overall strategy of each learning element is described and its contributions to the knowledge base are given.

7.2.1 Learning by abduction

A learning element performs abduction by selecting a clause from the knowledge base whose head is proven but whose body is yet to be proven and concludes the instance of the body. The instantiated elements of the body are added to the knowledge base.

Using the initial domain theory, in particular the clauses:

```

sister(X,Y) ← parent(Z,X),parent(Z,Y), female(X)
sister(jan, fred)

```

The unit clauses parent(patricia, jan), parent(patricia, fred) and female(jan) could be added to the knowledge base.

7.2.2 Learning by induction

An inductive learner could consider the following examples for father(X, Y):

- parent(graeme, grace), male(graeme), female(grace) (+)
- parent(graeme, jan), male(graeme), female(jan) (+)
- parent(fred, ann), male(fred), female(ann) (+)
- parent(patricia, fred), female(patricia), male(fred) (-)

Two plausible hypotheses that an inductive learning element might produce to explain these facts could be:

```

father(X,Y) ← male(X), parent(X, Y)
father(X,Y) ← female(Y), parent(X,Y)

```

7.2.3 Learning by discovery

A learning element could be defined to pro-

duce arbitrary clauses. This learning element might be constructed analogously to a classifier system (Wilson, 1987) where strong clauses (reproduction), mixed (crossover) and modified (mutation).

Assume that a learning element performs discovery and produces the clause:

$\text{mother}(X,Y) \leftarrow \text{parent}(X,Y)$

7.3 PAC-Error identification

Suppose the knowledge base has gained clauses from each of the learning elements described previously. It contains the following clauses:

- (C1) $\text{father}(X,Y) \leftarrow \text{male}(X), \text{parent}(X,Y)$
- (C2) $\text{father}(X,Y) \leftarrow \text{female}(X), \text{parent}(X,Y)$
- (C3) $\text{sister}(X,Y) \leftarrow \text{parent}(Z,X), \text{parent}(Z,Y), \text{female}(X)$
- (C4) $\text{mother}(X,Y) \leftarrow \text{parent}(X,Y)$
- (C5) $\text{sister}(\text{jan}, \text{fred}) \leftarrow$
- (C6) $\text{parent}(\text{patricia}, \text{jan}) \leftarrow$
- (C7) $\text{parent}(\text{patricia}, \text{grace}) \leftarrow$
- (C8) $\text{parent}(\text{patricia}, \text{fred}) \leftarrow$
- (C9) $\text{parent}(\text{fred}, \text{ann}) \leftarrow$
- (C10) $\text{parent}(\text{fred}, \text{william}) \leftarrow$
- (C11) $\text{parent}(\text{graeme}, \text{fred}) \leftarrow$
- (C12) $\text{male}(\text{fred}) \leftarrow$
- (C13) $\text{female}(\text{jan}) \leftarrow$

This knowledge base h entails the solutions given in Table 2 with respect to the goal clauses:

Goal Clause(g)	h_g
$\text{father}(\text{fred}, X)$	$\text{father}(\text{fred}, \text{ann})$ $\text{father}(\text{fred}, \text{william})$
$\text{father}(X, \text{jan})$	$\text{father}(\text{patricia}, \text{grace})$
$\text{mother}(X, \text{fred})$	$\text{mother}(\text{patricia}, \text{graeme})$
$\text{sister}(\text{jan}, X)$	$\text{sister}(\text{jan}, \text{grace})$ $\text{sister}(\text{jan}, \text{fred})$ $\text{sister}(\text{jan}, \text{jan})$

Table 2

The true standard deviation is:

$$\sigma = \sqrt{E(X^2) + \mu^2}$$

The mean error probability is the expected value of the error value which is:

$$\mu = E(\text{error}) = \frac{2}{7} \times 0.3 + \frac{1}{7} \times 0.3 + \frac{1}{7} \times 0.2 = 0.1571$$

The value of σ is 0.1. The sample size required to estimate the error with $d = 0.01$ and $\delta = 0.05$ is 385.

A typical execution of the ConfThresh algorithm would give the probabilities in Table 3

g instance	Refutation Set	TP	FP
$\text{father}(\text{fred}, \text{ann})$	C1, C9, C12	0.0285	0
$\text{father}(\text{fred}, \text{william})$	C1, C10, C12	0.0285	0
$\text{father}(\text{patricia}, \text{jan})$	C2, C6, C13	0	0.0428
$\text{mother}(\text{patricia}, \text{fred})$	C4, C8	0.0428	0
$\text{mother}(\text{graeme}, \text{fred})$	C4, C11	0	0.0428
$\text{sister}(\text{jan}, \text{grace})$	C3, C6, C7, C13	0.0285	0
$\text{sister}(\text{jan}, \text{fred})$	C3, C6, C8, C13	0.0285	0
$\text{sister}(\text{jan}, \text{jan})$	C3, C6, C13	0	0.0285

Table 3: Refutation Sets

and Table 4.

Clause	TP	FP
C1	0.057	0
C2	0	0.0428
C3	0.057	0.0285
C4	0.0428	0.0428
C5	0.0285	0
C6	0.057	0.0713
C7	0.0285	0
C8	0.0713	0
C9	0.0285	0
C10	0.0285	0
C11	0	0.0428
C12	0.057	0
C13	0.057	0.0713

Table 4: Clause Probabilities

The FN-probability for the hypothesis h is

0.0428 and the FP-probability is 0.1141. The total error is 0.1569. There are four candidates for deletion: C2, C6, C11 and C13. Each candidate has an FP-probability greater than the TP-probability. This means that if this clause were to be deleted, the increase in FN from TP is smaller than the decrease in FP. The error probability decreases.

Assume that C2 is chosen for deletion. It directly decreases FP with no effect on TP. FN remains the same but FP for the new hypothesis h_2 is reduced by the FP-probability of C2. The error probability for h_2 is $0.0428 + 0.0713 = 0.1141$.

There are two clauses affected by this deletion. Both C6 and C13 have their FP-probabilities reduced to 0.0285.

The second iteration has only one candidate for deletion: C11. It also has TP-probability of 0 so the error for h_3 becomes $0.0428 + 0.0285 = 0.713$.

No further clauses can be deleted. The hypothesis $h_3 = h - \{C2, C11\}$ is a local optimum.

8 Conclusion

This paper has several results:

1. It introduces multisource learning and shows that it can be used as a framework for multistrategy learning.
2. It develops a PAC characterization for function free definite clauses with respect to a population of goal clauses.
3. It introduces PAC-Error identification as a reformulation of PAC-learning more suitable for a multisource learning model.
4. It calculates appropriate sample sizes for obtaining confidence in hypotheses. Formulas are based on the central limit theorem from statistics.

5. It shows what probability regions exist across a universe of instances and how these regions respond to a learning algorithm.

Results using this model have been promising but are in the preliminary stages. It must be determined how successfully this model scales to a larger domain theory. Exploring various combinations of the number and types of learning elements would determine the generality of this model.

Acknowledgements

Thanks to Dr. V. Ciesielski and Dr. J Zobel for their invaluable comments on drafts of this paper.

References

- Kearns, M. J., *The Computational Complexity of Machine Learning*, The MIT Press, 1989.
- Lloyd J. W., *Foundations of Logic Programming*, Springer-Verlag 1984.
- Valiant L.G., A Theory of the Learnable, *Communications of the ACM*, 27(11), November 1984.
- Walpole R. E., Myers R.H., *Probability and Statistics for Engineers and Scientists*, Collier and MacMillan, 1978.
- Wilson S. W., Classifier Systems and the Animat Problem, *Machine Learning Journal*, 2(3), November 1987.

II. Knowledge Base Refinement

Symbolic Revision of Theories with M-of-N Rules *

Paul T. Baffes and Raymond J. Mooney

Department of Computer Sciences

University of Texas

Austin, TX 78712

baffes@cs.utexas.edu, mooney@cs.utexas.edu

Abstract

This paper presents a major revision of the EITHER propositional theory refinement system. Two issues are discussed. First, we show how run time efficiency can be greatly improved by changing from an exhaustive scheme for computing repairs to an iterative greedy method. Second, we show how to extend EITHER to refine M-of-N rules. The resulting algorithm, NEITHER (New EITHER), is more than an order of magnitude faster and produces significantly more accurate results with theories that fit the M-of-N format. To demonstrate the advantages of NEITHER, we present preliminary experimental results comparing it to EITHER and various other systems on refining the DNA promoter domain theory.

1 Introduction

Recently, a number of machine learning systems have been developed that use examples to revise an approximate (incomplete and/or incorrect) domain theory [Ginsberg, 1990; Ourston and Mooney, 1990; Towell and Shavlik, 1991; Danyluk, 1991; Whitehall *et al.*, 1991; Matwin and Plante, 1991]. Most of these systems revise theories composed of strict if-then rules (Horn clauses). However, many concepts are best represented using some form of partial matching or evidence summing, such as M-of-N concepts, which are true if at least M of a set of N specified features are present in an example.

*Supported by the NASA Graduate Student Researchers Program under grant number NGT-50732, the National Science Foundation under grant IRI-9102926, and a grant from the Texas Advanced Research Program under grant 003658144. This paper was originally published in the proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

There has been some work on the induction of M-of-N rules that demonstrates the advantages of this representation [Spackman, 1988; Murphy and Pazzani, 1991]. Other work has focused on revising rules that have real-valued weights [Towell and Shavlik, 1992; Mahoney and Mooney, 1992]. However, revising theories with simple M-of-N rules has not previously been addressed. Since M-of-N rules are more constrained than rules with real-valued weights, they provide a stronger bias and are easier to comprehend.

This paper presents a major revision of the EITHER propositional theory refinement system [Ourston and Mooney, 1990; Ourston and Mooney, in press] that is significantly more efficient and is also capable of revising theories with M-of-N rules. EITHER is inefficient because it computes a potentially exponential number of repairs for each failing example. The new version, NEITHER (New EITHER), computes only the single best repair for example, and is therefore much more efficient.

Also, because it was restricted to strict Horn-clause theories, EITHER did not produce as accurate results as KBANN (a neural-network revision system) on the DNA promoter problem [Towell and Shavlik, 1991; Towell and Shavlik, 1992]. Some aspects of the promoter concept fit the M-of-N format, since there are several potential sites where hydrogen bonds can form between the DNA and a protein; if enough of these bonds form, promoter activity can occur. EITHER attempts to learn this concept by forming a separate rule for each potential configuration by deleting different combinations of antecedents from the initial rules. Since a combinatoric number of such rules is needed to accurately model an M-of-N concept, the generality of the resulting theory is impaired. NEITHER, however, includes the ability to generalize a rule by lowering the threshold on an M-of-N rule. Including threshold changes as an alternative method for covering misclassified examples was

easily incorporated within the basic EITHER framework.

To demonstrate the advantages of NEITHER, we present experimental results comparing it to EITHER and various other systems on refining the promoter domain theory. NEITHER runs more than an order of magnitude faster than EITHER and produces a significantly more accurate theory with minor revisions that are easy to understand.

2 Theory Revision Algorithm

2.1 The EITHER Algorithm

The original EITHER theory refinement algorithm has been presented in various levels of detail in [Ourston and Mooney, 1990; Ourston and Mooney, in press; Ourston, 1991]. It was designed to repair propositional Horn-clause theories that are either overly-general or overly-specific or both. An overly-general theory is one that causes an example (called a *failing negative*) to be classified in categories other than its own. EITHER specializes existing antecedents, adds new antecedents, and retracts rules to fix these problems. An overly-specific theory causes an example (called a *failing positive*) not to be classified in its own category. EITHER retracts and generalizes existing antecedents and learns new rules to fix these problems. Unlike other theory revision systems that perform hill-climbing (and are therefore subject to local maxima), EITHER is guaranteed to fix any arbitrarily incorrect propositional Horn-clause theory [Ourston, 1991].

EITHER Main Loop

```

Compute all repairs for each example
While some examples remain uncovered
  Add best repair to cover set
  Remove examples covered by repair
end
Apply repairs in cover set to theory

```

NEITHER Main Loop

```

While some examples remain
  Compute a single repair for each example
  Apply best repair to theory
  Remove examples fixed by repair
end

```

Figure 1: Comparison of EITHER and NEITHER algorithms.

The algorithm used by EITHER for both generalization and specialization is shown in the top half of

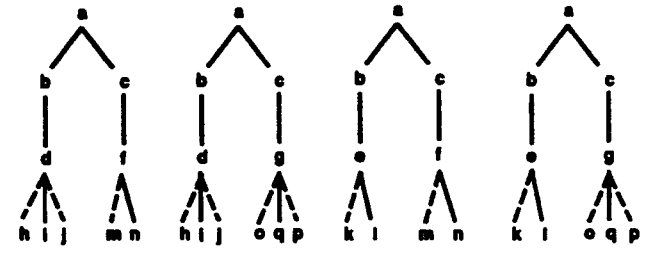
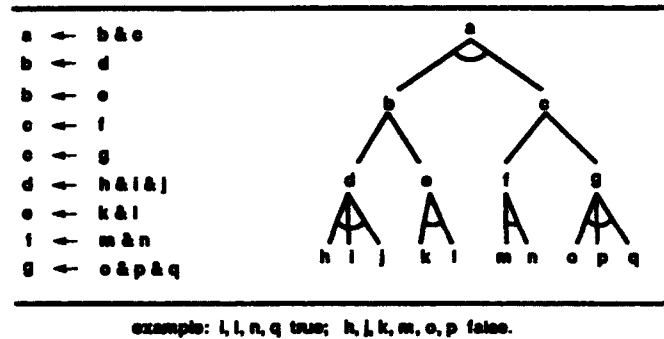


Figure 2: Partial proofs for unprovable positive example. Unprovable antecedents are shown with dotted lines.

Figure 1. There are three basic steps. First, *all* possible repairs for each failing example are computed. Next, EITHER enters a loop to compute a subset of these repairs that can be applied to the theory to fix all of the failing examples. This subset is called a *cover*. Repairs are ranked according to a benefit-to-cost ratio that trades off the number of examples covered against the size of the repair and the number of new failing examples it creates. The best repair is added to the cover on each iteration. Lastly, the repairs in the cover are applied to the theory. If the application of a repair over-compensates by creating new failing examples, EITHER passes the covered examples and the new failing examples to an induction component.¹ The results of the induction are added as a new rule when generalizing or as additional antecedents when specializing.

The time consuming part of this algorithm is the first step where all repairs for a given failing example are found. Figure 2 illustrates this process for theory generalization where EITHER is searching for leaf-rule² antecedent retractions to correct failing positive examples. The upper half of the diagram shows an input theory both as rules (on the left) and as

¹ EITHER uses a version of ID3 [Quinlan, 1986] for its induction.

² A leaf rule is a rule whose antecedents include an observable or an intermediate concept that is not the consequent of any existing rule.

Generalization					Specialization				
<i>change</i>	<i>resulting rule</i>	<i>b</i>	<i>c</i>	<i>bc</i>	<i>change</i>	<i>resulting rule</i>	<i>b</i>	<i>c</i>	<i>bc</i>
orig. rule	$a \leftarrow 2 \text{ of } (b, c)$	N	N	Y	orig. rule	$a \leftarrow 1 \text{ of } (b, c)$	Y	Y	Y
threshold -1	$a \leftarrow 1 \text{ of } (b, c)$	Y	Y	Y	threshold +1	$a \leftarrow 2 \text{ of } (b, c)$	N	N	Y
delete b	$a \leftarrow c$	N	Y	Y	delete rule	none	N	N	N

Table 1: Comparison of Revisions.

an AND-OR graph. The lower half of the diagram shows a hypothetical failing positive example and its partial proofs.³ From these proofs there are four possible repairs which will fix the example: retract h, j, m ; retract h, j, o, p ; retract k, m ; retract k, o, p . Theory specialization follows a similar process to return sets of leaf-rule retractions which fix individual failing negative examples.

2.2 Speeding Up EITHER

We have recently implemented a new version of EITHER (NEITHER) that takes a different approach, as shown in the bottom half of Figure 1. Two new algorithms form the basis for the difference between EITHER and NEITHER. First, calculation of repairs is now achieved in linear time. Second, all searches through the theory (for deduction, antecedent retraction and rule retraction) are optimized in NEITHER to operate in linear time by marking the theory to avoid redundant subproofs. NEITHER abandons the notion of searching for all partial proofs in favor of a greedy approach which rapidly selects a *single* best repair for each example. The three steps of the old EITHER algorithm can then be integrated into a single loop (see Figure 1).

To illustrate how repairs are computed in linear time, refer again to Figure 2. Rather than computing all partial proofs, NEITHER works bottom-up, constructing a single set of retractions. When multiple options exist, NEITHER alternates between returning the smallest option and returning the union of the options, depending whether the choice involves an AND or OR node. For generalization, retractions are unioned at AND nodes because all unprovable antecedents must be removed to make the rule provable. At OR nodes, only the smallest set of retractions is kept since only one rule need be provable. For specialization, these choices are reversed. Results are unioned at OR nodes to disable all rules which fire for a faulty concept. At AND nodes, the smallest set of rule retractions is selected since any single failure will disable a rule.

As an example, in Figure 2 the antecedent retrac-

tion calculations for the example would begin at the root of the graph, recursively calling nodes b and c . Retraction for node b then recurses on nodes d and e . When the recursion returns back to node b a choice must be made between the results from nodes d and e because node b is an OR node. Since the latter requires fewer retractions, it is chosen as the return value for node b . This process continues, resulting in a final repair: retract k, m .

Note that this algorithm is linear in the size of the theory. No node is visited more than once, and the computation for choosing among potential retractions must traverse the length of each rule at most once. The final repair is also minimum with respect to the various choices made along the way; it is not possible to find a smaller repair that will satisfy the example. This new algorithm thus trades the complete information available in the partial proofs for speed in computation.

2.3 Adding M-of-N Rules to NEITHER

With M-of-N rules, there are six types of revisions that can be made to a theory. As before, antecedents may be deleted or rules may be added to generalize the theory, and antecedents may be added or rules deleted to specialize the theory. The two new revisions are to increase or decrease the threshold: decreasing generalizes a rule and increasing specializes it.

To incorporate these two new revisions, NEITHER must be changed in four places. First, the computation of a repair for each failing example must take thresholds into account. For generalization, one need only retract enough antecedents to make the rule provable; there is no need to retract all false antecedents if the rule has a threshold. For example, if the rule for e in Figure 2 had a threshold of 1 there would be no need to retract k to prove this rule. A similar accounting for thresholds is required for computing rule deletions for specialization. Note that during generalization the threshold of each rule from which antecedents are retracted must be decreased by the number of antecedents retracted to account for the smaller size of the rule.

Second, NEITHER must compute threshold re-

³A partial proof is one in which some antecedents cannot be satisfied.

pairs. Calculating threshold changes can be done in conjunction with the computation of antecedent and rule deletion repairs since it is directly related to how many of antecedents of a rule are provable. For generalization, we change the threshold to the number of antecedents which are provable. In specialization, we set the threshold to one more than the number of provable antecedents.

Third, a mechanism must be provided for selecting between a threshold change and a deletion. Effectively, this amounts to deciding which type of revision to try first. The philosophy used in NEITHER is to try the most aggressive changes initially in the hopes that the resulting repair will cover more examples. If the repair creates new failing examples, the less ambitious repairs are tried in turn with induction used as a last resort. During generalization, more radical repairs are those which create more general rules (i.e., rules which can prove more examples). In specialization, the opposite is true. As with EITHER, if all changes result in new failing examples, the algorithm falls back to induction to learn new rules or add new antecedents.

Table 1 compares equivalent threshold and deletion changes for generalization and specialization. The columns labeled with b, c and bc indicate whether the corresponding rule will conclude a when just b, just c or both b and c are true. Note that in both cases, the threshold change results in a more general rule. This means that threshold changes should be tried before antecedent deletions during generalization, but tried after rule deletions during specialization.

Fourth and finally, the induction component of NEITHER must be altered slightly to accommodate threshold rules. When the application of a repair causes new failing examples to occur, NEITHER resorts to induction as did EITHER. The result of the induction cannot, however, simply be added to the theory as before. Table 2 illustrates the problem. The original rule shown can be used to prove both the positive and negative examples, and deleting this rule or incrementing its threshold only prevents the positive example from being proved. Assume that induction returns a new feature, d, which can be used to distinguish the two examples (i.e., d is true for the positive example but false for the negative example). Because the original rule has a threshold, adding d directly will still allow both examples to prove the rule. This problem remains even if one tries to increment the threshold in addition to adding d. Instead, the rule must be *split* by renaming the consequent of the original rule, and creating a new rule with the renamed consequent and the re-

<i>example features</i>	<i>pos. example</i>	<i>neg. example</i>
b, c, d	b, \neg c, d	b, c, \neg d
<i>orig. rule</i>	<i>pos. example</i>	<i>neg. example</i>
a \leftarrow 1 of (b,c)	Y	Y
<i>add to rule</i>	<i>pos. example</i>	<i>neg. example</i>
a \leftarrow 1 of (b,c,d)	Y	Y
<i>split rule</i>	<i>pos. example</i>	<i>neg. example</i>
X \leftarrow 1 of (b,c)	Y	Y
a \leftarrow X,d	Y	N

Table 2: Induced Antecedent Addition.

sults of induction as the new rule's antecedent list.

3 Experimental Results

3.1 Experimental Design

For the purposes of this paper, the resulting algorithm is labeled NEITHER-MOFN. We tested both NEITHER and NEITHER-MOFN against other classification algorithms using the DNA promoter sequences data set [Towell *et al.*, 1990]. This data set involves 57 features, 106 examples, and 2 categories. The theory provided with the data set has an initial classification accuracy of 50%. We selected this particular data set because EITHER performed poorly on data sets best modelled using M-of-N rules. In addition to testing EITHER, NEITHER and NEITHER-MOFN, we ran experiments using ID3 [Quinlan, 1986], backpropagation [Rumelhart *et al.*, 1986] and RAPTURE [Mahoney and Mooney, in press] (a revision system based on certainty factors).

The experiments proceeded as follows. Each data set was divided into training and test sets. Training sets were further divided into subsets, so that the algorithms could be evaluated with varying amounts of training data. After training, each system's accuracy was recorded on the test set. To reduce statistical fluctuations, the results of this process of dividing the examples, training, and testing were averaged over 25 runs. The random seeds for the backpropagation algorithm were reset for each run. Training time, and test set accuracy were recorded for each run. Statistical significance was measured using a Student t-test for paired difference of means at the 0.05 level of confidence (i.e., 95% certainty that the differences are not due to random chance).

3.2 Results

The results of our experiments are shown in the three graphs of Figures 3, 4 and 5. Figure 3 compares the learning curves of the systems tested, showing how predictive accuracy on the test set changes

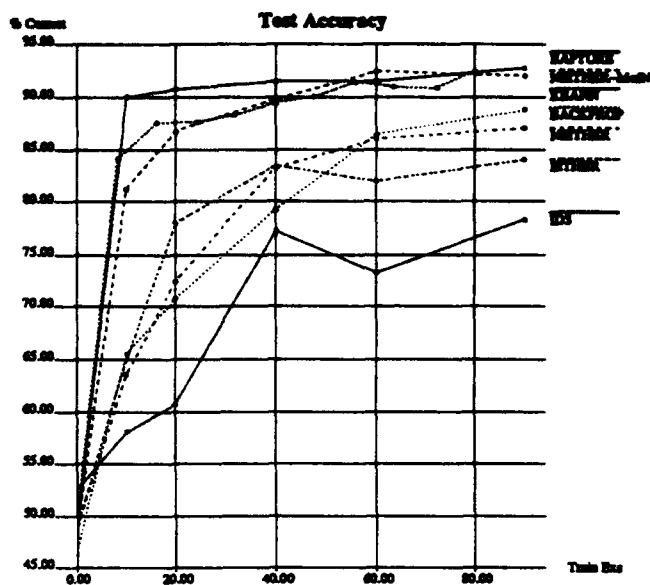


Figure 3: Test Set Accuracy

as a function of the number of training examples. As can be seen NEITHER-MOFN's performance was significantly better than all other systems except RAPTURE and KBANN.⁴ RAPTURE out-performed NEITHER-MOFN with small numbers of training examples but their accuracy was comparable with larger inputs. NEITHER's accuracy was on par with backpropagation, but was lower than EITHER for small training sets and higher than EITHER for large training sets. Note, that Figure 3 is not direct comparison of NEITHER and KBANN since the results reported were compiled from different subsets of the DNA promoter sequences data set. ID3 had significantly lower accuracy than the other systems.

Figure 4 shows a comparison of training times. Both NEITHER-MOFN and NEITHER were more than an order of magnitude faster than backpropagation and EITHER. Only ID3 ran faster than NEITHER-MOFN.

We also collected data on the average complexity of the revised theories produced by both NEITHER and NEITHER-MOFN. Complexity was measured as the total size; i.e., the total number of all literals in the theory. The results are shown in Figure 5. As can be seen from this graph, NEITHER-MOFN not only produces less complex resulting theories but also produces theories closer in size to the original.

⁴Technically, the last difference between backpropagation and NEITHER-MOFN was only significant at the 0.1 level.

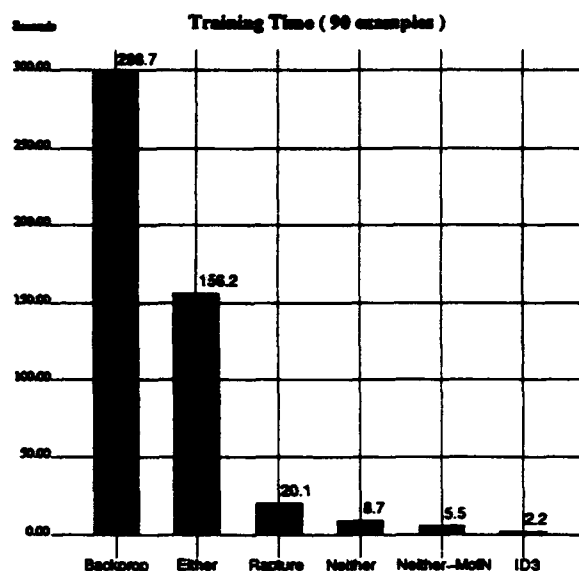


Figure 4: Training Time Comparison

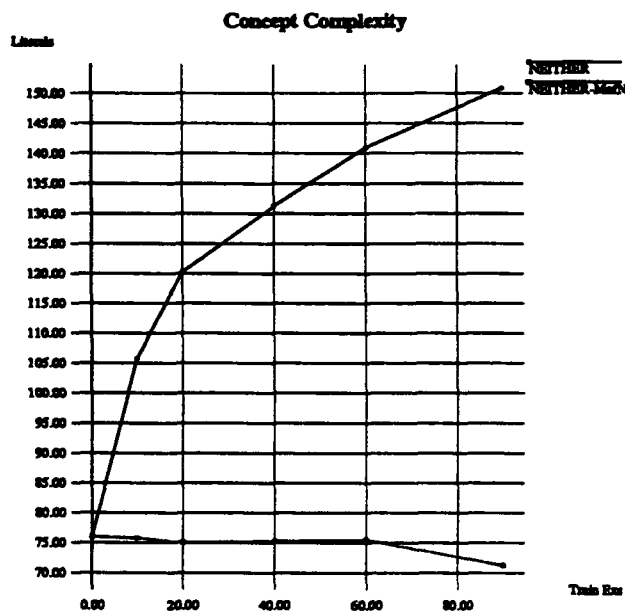


Figure 5: Concept Complexity

3.3 Discussion

Many of our expectations were borne out by the experimental results. Both NEITHER and NEITHER-MOFN ran more than an order of magnitude faster than EITHER due to the optimized algorithms discussed in section 2. NEITHER-MOFN's increase in accuracy was also expected since the new algorithm is able to concentrate on making M-of-N revisions directly. Also, the fact that NEITHER-MOFN generates less complex theories is not surprising, again because it can directly modify threshold values rather than create new rules. In short, by adding one more operator to the generalization and specialization processes, NEITHER-MOFN is able to accurately revise a theory known to be difficult for symbolic systems, without having to sacrifice the efficiency of a symbolic approach. Finally, the most comparable learning-curve results from [Towell, 1991] would indicate that KBANN's accuracy in the promoter domain is about the same as NEITHER-MOFN's.

4 Related Work

Several researchers have developed methods for inducing M-of-N concepts from scratch. CRLS [Spackman, 1988] learns M-of-N rules and out-performed standard rule induction in several medical domains. ID-2-of-3 [Murphy and Pazzani, 1991] incorporates M-of-N tests in decision-tree learning and out-performed standard decision-tree induction in a number of domains. Both projects clearly demonstrate the advantages of M-of-N rules.

SEEK2 [Ginsberg *et al.*, 1988] includes operators for refining M-of-N rules; however, its revision process is heuristic and it is not guaranteed to produce a revised theory that is consistent with all of the training examples. NEITHER uses a greedy covering approach to guarantee that it finds a set of revisions that fix all of the misclassified examples in the training set. Also, unlike NEITHER, SEEK2 cannot learn new rules or add new antecedents to existing rules.

KBANN [Towell and Shavlik, 1992] revises a theory by translating it into a neural network, using backpropagation to refine the weights, and then retranslating the result back into symbolic rules. NEITHER's symbolic revision process is much more direct and, from all indications, significantly faster. Although KBANN's results are referred to as M-of-N rules, they actually contain real-valued antecedent weights and therefore are not strictly M-of-N. In addition, KBANN's revised theories for the promoter problem are also more complex in terms of number of antecedents than the initial theory [Towell, 1991],

while NEITHER actually produces a slight reduction. Therefore, NEITHER's revised theories are less complex and presumably easier to understand. Finally, unlike KBANN, NEITHER is guaranteed to converge to 100% accuracy on the training data.

RAPTURE [Mahoney and Mooney, 1992] uses a combination of symbolic and neural-network learning methods to revise a certainty-factor rulebase [Buchanan and E.H. Shortliffe, 1984]. Consequently, it lies somewhere between NEITHER and KBANN on the symbolic-connectionist dimension. As illustrated in the results, its accuracy on the promoter problem is only slightly superior to NEITHER's. However, its real-valued certainty factors make its rules more complex.

5 Future Work

The current version of NEITHER needs to be enhanced to handle a number of issues. We need to incorporate a number of advanced features from EITHER, such as constructive induction, modification of higher-level rules, and the ability to handle numerical features and noisy data. Also, we could to extend our methods to handle negation as failure and incorporate the ability to handle M-of-N rules into first-order theory revision [Richards and Mooney, 1991]. Finally, we need to perform a more comprehensive experimental evaluation of the system.

6 Conclusions

This paper has presented an efficient propositional theory refinement system that is capable of revising M-of-N rules. The basic framework is a modification of EITHER [Ourston and Mooney, 1990]; however, the construction of partial proofs has been reduced from exponential to linear time and a method for revising the thresholds of M-of-N rules has been incorporated. The resulting system runs more than an order of magnitude faster and produces significantly more accurate results in domains requiring partial matching, such as the problem of recognizing promoters in DNA.

Acknowledgements

Special thanks to Chris Whatley for his help implementing NEITHER.

References

- [Buchanan and E.H. Shortliffe, 1984]
G.G. Buchanan and eds. E.H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.*

- Addison-Wesley Publishing Co., Reading, MA, 1984.
- [Danyluk, 1991] A. D. Danyluk. Gemini: An integration of analytical and empirical learning. In *Proceedings of the International Workshop on Multistrategy Learning*, pages 191–206, Harper's Ferry, W.Va., Nov. 1991.
- [Ginsberg et al., 1988] A. Ginsberg, S. M. Weiss, and P. Politakis. Automatic knowledge based refinement for classification systems. *Artificial Intelligence*, 35:197–226, 1988.
- [Ginsberg, 1990] A. Ginsberg. Theory reduction, theory revision, and retranslation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 777–782, Detroit, MI, July 1990.
- [Mahoney and Mooney, 1992] J. Mahoney and R. Mooney. Combining symbolic and neural learning to revise probabilistic theories. In *Proceedings of the ML92 Workshop on Integrated Learning in Real Domains*, Aberdeen, Scotland, July 1992.
- [Mahoney and Mooney, in press] J. J. Mahoney and R. J. Mooney. Combining neural and symbolic learning to revise probabilistic rule bases. In *Advances in Neural Information Processing Systems*, Vol. 5, San Mateo, CA, in press. Morgan Kaufman.
- [Matwin and Plante, 1991] S. Matwin and B. Plante. A deductive-inductive method for theory revision. In *Proceedings of the International Workshop on Multistrategy Learning*, pages 160–174, Harper's Ferry, W.Va., Nov. 1991.
- [Murphy and Pazzani, 1991] P. M. Murphy and M. J. Pazzani. ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 183–187, Evanston, IL, June 1991.
- [Ourston and Mooney, 1990] D. Ourston and R. Mooney. Changing the rules: a comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815–820, Detroit, MI, July 1990.
- [Ourston and Mooney, in press] D. Ourston and R. J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, in press.
- [Ourston, 1991] D. Ourston. *Using Explanation-Based and Empirical Methods in Theory Revision*. PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, August 1991.
- [Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [Richards and Mooney, 1991] B. Richards and R. Mooney. First-order theory revision. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 447–451, Evanston, IL, June 1991.
- [Rumelhart et al., 1986] D. E. Rumelhart, G. E. Hinton, and J. R. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, Vol. I, pages 318–362. MIT Press, Cambridge, MA, 1986.
- [Spackman, 1988] K. A. Spackman. Learning categorical decision criteria in biomedical domains. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 36–46, Ann Arbor, MI, June 1988.
- [Towell and Shavlik, 1991] G. Towell and J. Shavlik. Refining symbolic knowledge using neural networks. In *Proceedings of the International Workshop on Multistrategy Learning*, pages 257–272, Harper's Ferry, W.Va., Nov. 1991.
- [Towell and Shavlik, 1992] G. Towell and J. Shavlik. Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann, 1992.
- [Towell et al., 1990] G. G. Towell, J. W. Shavlik, and Michiel O. Noordewier. Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, MA, July 1990.
- [Towell, 1991] G. G. Towell. *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction*. PhD thesis, University of Wisconsin, Madison, WI, 1991.
- [Whitehall et al., 1991] B. L. Whitehall, S. C. Lu, and R. E. Stepp. Theory completion using knowledge-based learning. In *Proceedings of the International Workshop on Multistrategy Learning*, pages 144–159, Harper's Ferry, W.Va., Nov. 1991.

Knowledge Base Refinement Through a Supervised Validation of Plausible Reasoning

Gheorghe Tecuci
AI Center, CS Department
George Mason University
4400 Univ. Dr, Fairfax, VA 22030, USA
and Romanian Academy, Romania
tecuci@aic.gmu.edu

David Duff
AI Center
George Mason University
4400 Univ. Dr, Fairfax, VA 22030, USA
and The MITRE Corp. AI Technical Center
duff@mitre.org

Abstract

This paper presents an integrated heuristic approach to knowledge base refinement which is viewed as a supervised validation of plausible reasoning. The approach integrates multistrategy learning based on multitype inference, active experimentation, and guided knowledge elicitation. One of the main features of this approach is that once the knowledge base has been refined to deductively entail a new piece of knowledge, it can be easily further refined to deductively entail many other similar pieces of knowledge.

Keywords: multistrategy learning, knowledge acquisition, plausible reasoning

1. Introduction

An expert system consisting of an incomplete and partially incorrect knowledge base (KB), and of a deductive inference engine, suffers from two major limitations:

- it is not able to solve some problems from its domain of expertise (because the KB is incomplete);
- the solutions proposed might be incorrect (because the KB is partially incorrect).

The set of problems which such a system could solve is the *deductive closure* of the knowledge base (DC). In the case of a theorem prover, it is the set of facts which could be deductively inferred from the KB.

That is, $DC = \{I : KB \models I\}$

where " \models " means deductive entailment.

Figure 1 shows the relationship between the deductive closure DC of the imperfect KB of an expert system and the set of true facts in the application domain (TC):

- DC and TC are "crisp" sets, with clearly defined borders. That is, the system has an algorithm for testing the membership of a statement in DC , and presumably a human expert can perform a similar test on TC .
- $DC \cap TC$ represents the set of facts which are deductively entailed by the KB and are true. This shows that there is useful and correct knowledge encoded into the facts and the deductive rules of the KB.
- $DC - TC$ represents the set of facts which are deductively entailed by the KB but are false. This shows that there are errors in the set of facts and deductive rules.
- $TC - DC$ represents the set of facts which are true but are not deductively entailed by the KB. This shows that the set of facts and deductive rules is incomplete.

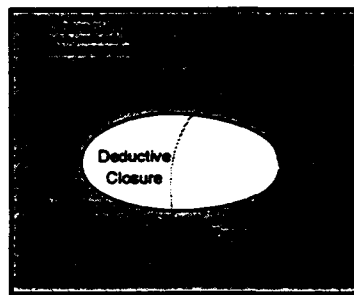


Figure 1: The relationship between
 DC and TC .

The goal of KB refinement is to improve the

knowledge base so that DC becomes a good approximation of TC . As a result, the KB would become an almost complete and correct one, and the expert system would be able to correctly solve most of the problems from its domain of expertise.

Many of the current KB refinement systems such as ANA-EBL (Cohen, 1991), CLINT (De Raedt and Bruynooghe, 1993), DUCTOR (Cain, 1991), EITHER (Mooney and Ourston, 1993), FORTE (Richards and Mooney, 1993), SEEK (Ginsberg, Weiss, and Politakis, 1988), try to partially generalize the KB so as to cover more of TC , and to partially specialize it, so as cover less of $DC - TC$. In the case of a Prolog-like KB, this is accomplished by generalizing and/or specializing some of the rules, as well as by introducing new facts into the KB, and/or removing other facts.

In this paper, we are also addressing the problem of correcting and extending DC so as to better approximate TC . However, our approach, as opposed to the approaches cited above, brings a new set into play, the plausible closure of a KB, and proposes a different perspective to the KB refinement problem.

2. The plausible closure of the KB

We are assuming that the initial incomplete and partially incorrect KB consists of facts and rules expressed in first-order logic. However, the rules are not restricted to be deductive. They might also be weaker correlations as determinations (Davies and Russell, 1987; Russell, 1989), mutual dependencies (Michalski, 1993), etc. This is so for allowing the introduction of all sorts of relevant knowledge into the KB.

The *plausible closure* of the KB (PC) is defined as the set of problems which a *plausible inference engine* could solve. In the case of a theorem prover, it is the set of facts which could be plausibly inferred from the KB.

That is, $PC = \{I: KB \models I\}$

where " \models " means plausible entailment.

One way to make plausible inferences is to use the rules from the KB not only deductively, but also abductively or analogically.

Let us consider, for instance, the rule

$$\forall_x [P(x) \rightarrow Q(x)].$$

If one knows that $P(a)$ is true, then one may *deductively* infer $Q(a)$:

$$\{P(a), \forall_x [P(x) \rightarrow Q(x)]\} \models Q(a)$$

If one knows that $Q(a)$ is true, then one may *abductively* (Pople, 1973; Josephson, 1991) infer $P(a)$:

$$\{Q(a), \forall_x [P(x) \rightarrow Q(x)]\} \models P(a)$$

If one knows that $P(a)$ is true, and b is similar to a , then one may *analogically* (Carbonell, 1986; Gentner, 1990; Kedar-Cabelli, 1988; Kodratoff, 1990; Winston, 1980) infer $Q(b)$:

$$\left\{ \begin{array}{l} P(a), \\ \forall_x [P(x) \rightarrow Q(x)], \\ ("b \text{ 'similar' to } a") \end{array} \right\} \models Q(b)$$

Another way to make plausible inferences is to use weaker correlations between knowledge pieces (e.g. related facts, determinations, dependencies, "A is like B" statements, etc.).

Let us consider, for instance, that the KB contains the following related facts (each set describing an object):

$$P(c) \wedge Q(c) \wedge R(c)$$

$$P(d) \wedge Q(d) \wedge S(d)$$

$$P(e) \wedge Q(e)$$

Then one might empirically generalize (Mitchell, 1978; Michalski, 1983; Quinlan, 1986) these sets of facts to the rule

$$\forall_x [P(x) \rightarrow Q(x)]$$

and might deductively use this rule with the fact $P(a)$ to predict that $Q(a)$ is also true.

Analogical inferences could be made by employing plausible determinations (Russell, 1989; Tecuci, 1993). Let us consider, for instance, the following determination rule stating that U plausibly determines V :

$$U(x, y) \succ V(x, z)$$

Then one may make the following analogical inference:

$$U(s, a) \wedge V(s, b) \wedge U(t, a) \models V(t, b)$$

Another example of plausible inference is the "useful analogical inference", introduced by Greiner (1988). Let us suppose, for instance, that the system is told that $Q(b)$ is true, and is given the analogical hint " b is like a ", in order to show that $KB \models Q(b)$. That is, without this analogical hint, the system is not able to show that $KB \models Q(b)$. Based on this analogical hint, the system is looking for a feature of a (e.g. $P(a)$) which, if possessed by b , would allow it to prove $KB \models Q(b)$:

$$KB \models P(a)$$

$$KB \not\models P(b)$$

$$KB \not\models \neg P(b)$$

$$\{P(b), KB\} \models Q(b)$$

As a result of this reasoning $P(b)$ is asserted into the KB.

Several other types of plausible derivations based on implications and dependencies are described in (Collins and Michalski, 1989).

In order to show that a certain fact, I , is plausibly entailed by the KB, a system is not restricted to making only one plausible inference. In general, it could build a plausible justification tree (Tecuci, 1993). A plausible justification tree is like a proof tree, except that the inferences which compose it may be the result of different types of reasoning (not only deductive, but also analogical, abductive, predictive, etc.). An example of such a tree is presented in Figure 5.

One of the main reasons for illustrating the above plausible inferences was to show that, by employing a plausible inference engine, one could significantly extend the set of problems that could be solved by a system.

Figure 2 presents our conjecture about the relationships between the plausible closure of the KB, the deductive closure of the KB, and the set of true facts in the application domain:

- PC is a "soft" set, the boundaries of which are not strictly defined. Indeed, depending of the number and of the strength of the different types of plausible reasoning steps in a justification tree for a fact F , the plausibility of F is higher or lower.
- $PC \supset DC$ because the deductive proof trees are special cases of plausible

justification trees.

- $PC \cap TC$ represents the set of facts which are plausibly entailed by the KB and are true.
- $PC \cap TC - DC$ represents the set of true facts which are plausibly entailed by the KB, but are not deductively entailed by the KB. Our hypothesis is that this is a significantly large set.
- $TC - PC$ represents the set of true facts that are not plausibly entailed by the KB. Although this set is not well defined, it expresses the intuition that there are true facts which even a plausible inference engine could not derive from the current KB.

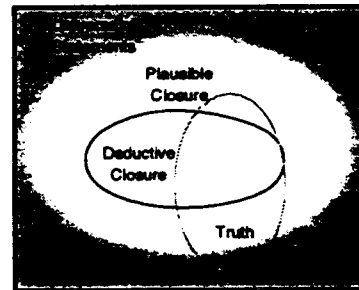


Figure 2: The relationship between DC , PC , and TC .

The deductive and plausible closures are two approximations of truth. In the approach we are proposing, we are considering DC as being an approximate lower bound for TC , and PC as being an approximate upper bound for TC . With this interpretation, the KB refinement problem reduces to one of determining the set TC in the plausible space defined by DC and PC . More precisely, during KB refinement, DC will be extended with a significant portion of $PC \cap TC$, and will also be corrected to remove from it most of $DC - TC$. Consequently, as a result of this process, DC will become a good approximation of TC .

Otherwise stated, we propose an approach to KB refinement which is viewed as a transfer of knowledge from the plausible closure to the deductive closure.

In this paper we are proposing a heuristic method which is an effective way of extending DC with a significant portion of

$PC \cap TC$. More precisely, during knowledge refinement, the sets DC and PC are transformed as follows:

- DC is extended by acquiring new facts or rules, or by generalizing some of the rules;
- DC is improved by specializing some of the deductive rules which could be partially incorrect;
- PC is extended and/or improved by acquiring new facts or rules, or improving some of the existing rules.

The next section contains a general presentation of the proposed KB refinement method.

3. General presentation of the KB refinement method

The KB of the system is assumed to be incomplete and partially incorrect. The KB is improved during training sessions with a human expert who provides the system with new input information I . Each such input I is an example of an answer that the final expert system should be able to generate, that is, I should be in the deductive closure of the final expert system. The goal of KB refinement is to improve the KB of the system so that to answer questions as the human expert.

If an input I is already in the plausible closure of the KB, then the system will be able to make a significant transfer of knowledge from the plausible closure to the deductive closure. More precisely, it will extend the de-

ductive closure with new hypotheses, H , so as to include a generalization I_g of I , that is

$$\{H, KB\} \models I_g$$

At the same time, it will extend the plausible closure of the KB, so as to include more of TC , and might also remove some inconsistencies from the deductive closure, reducing the size of the set $DC - TC$.

If I is not in the plausible closure of the KB, then it will be simply asserted into the KB. This has the effect of extending both DC and PC . Indeed, the presence of I in DC may make it possible for the system to show that other facts (e.g. I_1, I_2) are deductively or plausibly entailed by the KB:

$$KB \not\models I_1, \text{ but } \{I, KB\} \models I_1$$

$$KB \not\models I_2, \text{ but } \{I, KB\} \models I_2$$

This also shows that during KB refinement, PC may grow to include facts from $TC - PC$.

The main stages of the KB refinement process are presented in Figure 3. They are:

- multitype inference and generalization;
- experimentation, verification and repair;
- goal-driven knowledge elicitation.

In the first stage, the system analyzes the input in terms of its current knowledge by building a plausible justification tree which demonstrates that the input is a plausible consequence of the system's current knowledge.

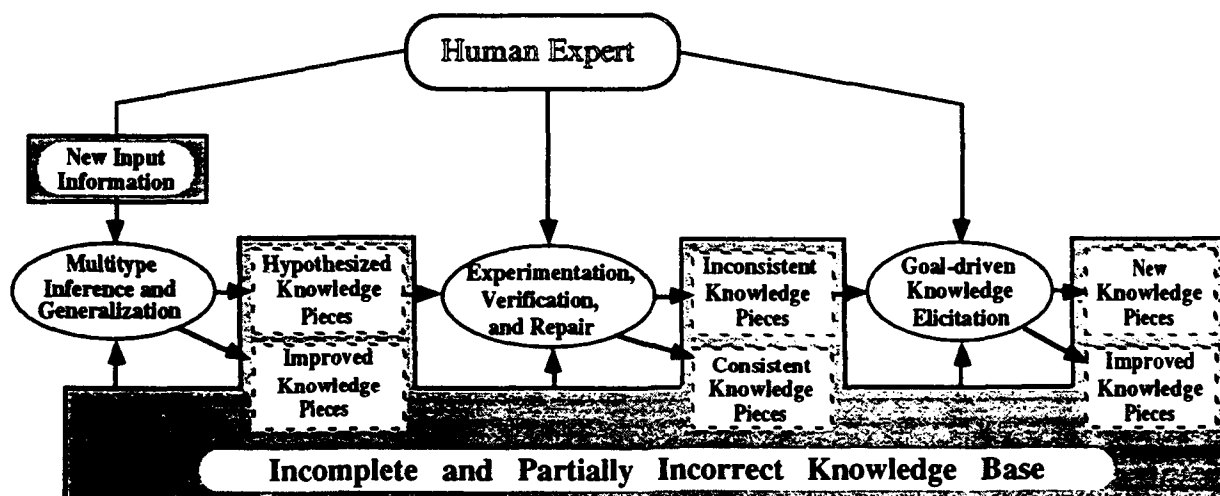


Figure 3: The main stages of the KB refinement process

As a result of the analysis of the input via multitype inferences, new pieces of knowledge are hypothesized (through analogy, abduction, inductive generalization and prediction, etc.), and existing pieces of knowledge are improved so that the extended knowledge base to entail the input. By asserting these pieces of knowledge into the knowledge base, the system is able to deductively entail the input. The support for these new pieces of knowledge is that they allow building a logical connection (the justification tree) between a knowledge base that represents a part of the real world, and a piece of knowledge (the input) that is known to be true in the real world.

Next, the system will generalize the plausible justification tree, by employing different types of generalizations (not only deductive or empirical, but also based on analogy and, possibly, on other types of inferences). By this, it will generalize the hypothesized knowledge, so that the resulting knowledge base will entail not only the received input I , but also a generalization of it I_g .

The generalized plausible justification tree shows how I_g (a generalization of the input I) is entailed by the KB. However, this tree was obtained by making both plausible inferences and plausible generalizations. Consequently, both the tree and the corresponding knowledge pieces learned are less certain. One may improve them by performing experiments. This is the second stage of the KB refinement process. During this stage, the system will generate instances of I_g and will ask the user if they are true or false. It will further improve the hypothesized knowledge pieces so that the updated KB to deductively entail the instances of I_g which are true and to reject the ones which are false.

However, because the KB is incomplete and possibly partially incorrect, some of the learned knowledge pieces may be inconsistent (i.e. may cover negative examples). In order to remove such inconsistencies, additional knowledge pieces (which represent new terms in the representation language of the system) are elicited from the expert, through several consistency-driven knowledge elicitation techniques. This represents the third stage of the KB refinement process.

The entire knowledge refinement process is characterized by a cooperation between the learning system and the human expert in which the learner performs most of the tasks and the expert helps it in solving the problems that are intrinsically difficult for a learner (e.g., the credit/blame assignment problem, the problem of new terms) and relatively easy for the human expert.

4. Exemplary application domain

We will use the domain of workstation allocation and configuration in order to illustrate this KB refinement method. The expert system to be built has to reason about which machines are suitable for which tasks and to allocate an appropriate machine for each task.

The initial (incomplete and partially incorrect) knowledge base contains information about various printers and workstations distributed throughout the workplace. A sample of this knowledge base is presented in Figure 4. Notice that it contains different types of knowledge: deductive rules, a plausible determination (Russell, 1989; Tecuci, 1993), facts, and hierarchies. Each of these knowledge pieces might be incomplete and/or partially incorrect.

Let us suppose that the system is told that `macII02` is suitable for publishing

`suitable(macII02, publishing)`

and this fact is representative of the type of answers it should be able to provide.

5. Multitype inference and generalization

The system tries to analyze ("understand") the input in terms of its current knowledge by building the plausible justification tree in Figure 5. Such a tree demonstrates that the input is a plausible consequence of the system's current knowledge. The method for building such a tree is presented in (Tecuci, 1993). It employs a backward chaining uniform-cost search.

The tree in Figure 5 is composed of four deductions, an inductive prediction, and a determination-based analogical implication.

```

suitable(X, publishing) :-
    runs(X, publishing-sw), communicate(X, Y),
    isa(Y, high-quality-printer).
; X is suitable for publishing if it runs
; publishing software and communicates
; with a high quality printer

communicate(X, Y) :-
    on(X, Z), on(Y, Z).
; X and Y communicate if they are on the same network

communicate(X, Y) :-
    on(X, Z), on(Y, V), connect(Z, V).
; X and Y communicate if they are on connected networks

isa(X, high-quality-printer) :-
    isa(X, printer), speed(X, high), resolution(X, high).
; X is a high quality printer if
; it has high speed and resolution

runs(X, Y) :- os(X, Z).
; the type of software which a machine could run is largely determined
; by its operating system (":-" means plausible determination)

runs(X, Y) :- runs(X, Z), isa(Z, Y).

os(sun01, unix).    on(sun01, fddi).    speed(sun01, high).    processor(sun01, risc).
; sun01's operating system is unix, it is on the fddi network, has high speed and a risc processor

os(hp05, unix).    on(hp05, ethernet).    speed(hp05, high).    processor(hp05, risc).    runs(hp05, frame-maker).
os(macplus07, mac-os).    on(macplus07, appletalk).
os(macII02, mac-os).    on(macII02, appletalk).
os(macIIc03, mac-os).    runs(macIIc03, page-maker).
on(proprinter01, ethernet).    resolution(proprinter01, high).    processor(proprinter, risc).
on(laserjet01, fddi).    resolution(laserjet01, high).    processor(laserjet01, risc).
on(microlaser03, ethernet).    resolution(microlaser03, high).    processor(microlaser03, risc).
resolution(xerox01, high).    speed(xerox01, high).    processor(xerox01, risc).
connect(appletalk, ethernet).    connect(appletalk, fddi).    connect(fddi, ethernet).

```

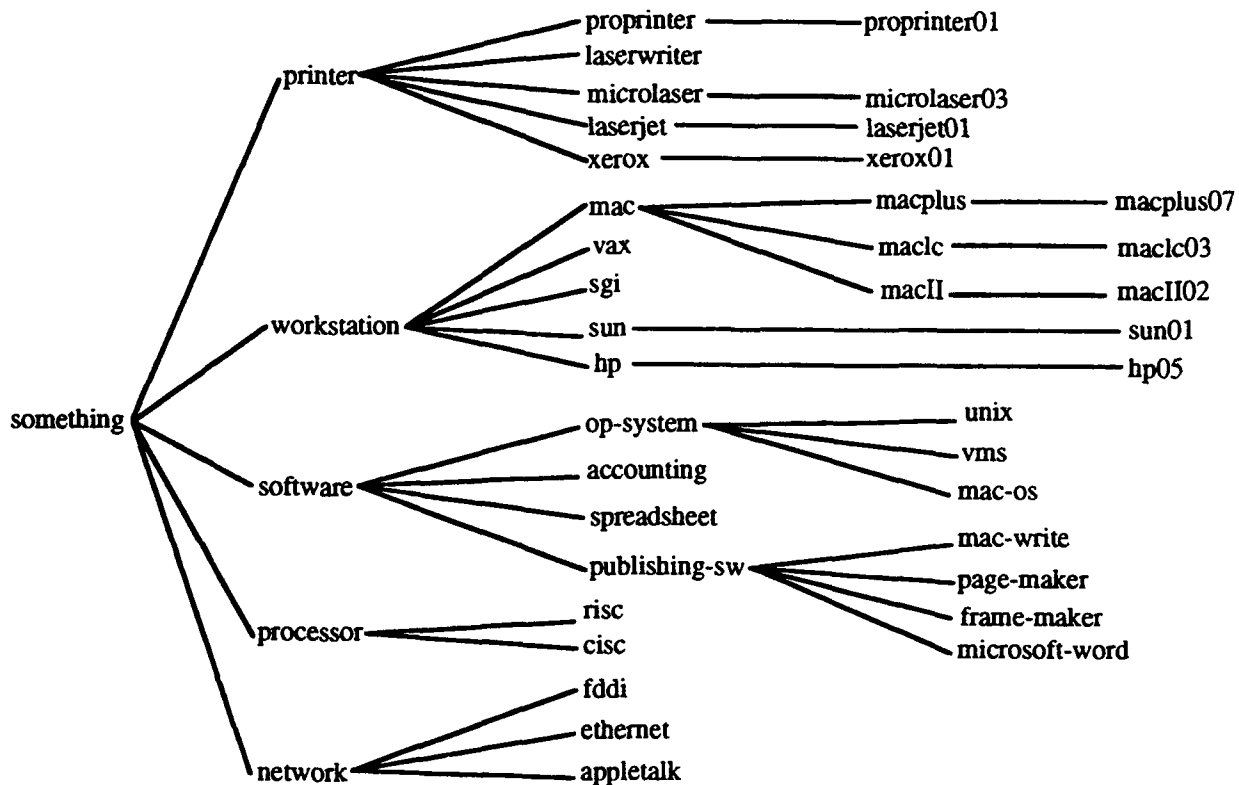


Figure 4: Sample of an incomplete and partially incorrect KB for the domain of workstation allocation and configuration.

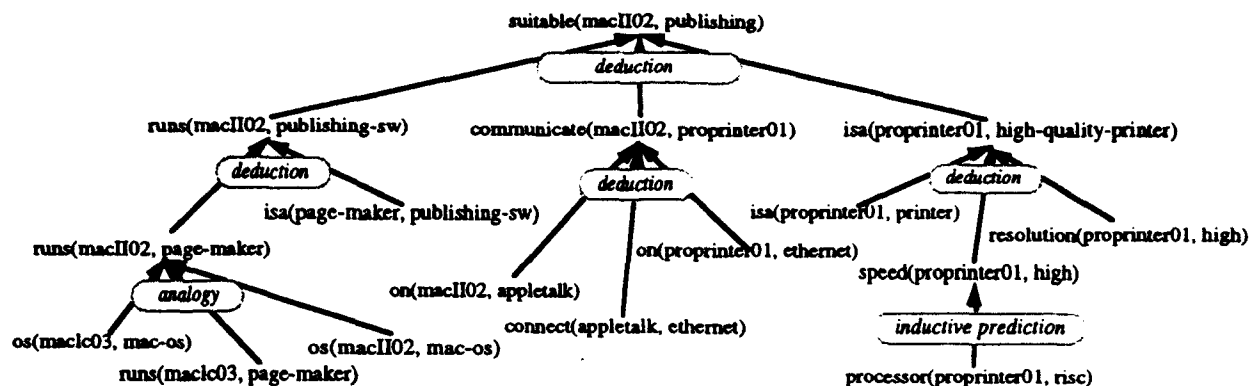


Figure 5: A plausible justification tree for "suitable(macII02, publishing)".

The inductive prediction

$\text{processor}(\text{proprinter01}, \text{risc}) \rightarrow \text{speed}(\text{proprinter01}, \text{high})$ was made by empirically generalizing the facts:

$\text{speed}(\text{sun01}, \text{high}), \text{os}(\text{sun01}, \text{unix}), \text{on}(\text{sun01}, \text{fdi}),$
 $\text{processor}(\text{sun01}, \text{risc}).$

$\text{speed}(\text{hp05}, \text{high}), \text{os}(\text{hp05}, \text{unix}), \text{on}(\text{hp05}, \text{ethernet}),$
 $\text{processor}(\text{hp05}, \text{risc}), \text{runs}(\text{hp05}, \text{frame-maker}).$

$\text{speed}(\text{xerox01}, \text{high}), \text{resolution}(\text{xerox01}, \text{high}),$
 $\text{processor}(\text{xerox01}, \text{risc}).$

to the rule

$\text{speed}(x, \text{high}) :- \text{processor}(x, \text{risc})$

and then applying this rule deductively.

An open problem is how to collect the facts to be generalized, and what kind of generalization to look for. One solution which we are investigating is based on CLINT's approach (De Raedt, 1991) of using a hierarchy of languages for the rules to be learned. Each language is characterized by a certain form of the rules to be learned, which suggests the kind of facts to look for.

The analogical inference was made by using the determination rule

$\text{runs}(X, Y) :- \text{os}(X, Z)$

as indicated in Figure 6.

While there may be several justification trees for a given input, the attempt is to find the most simple and the most plausible one (Lee, 1993). This tree shows how a true fact I derives from other true facts from the KB. Based on the Occam's razor (Blumer, Ehrenfeucht, Haussler, and Warmuth, 1987), and of the general hypothesis used in abduction which states that the best explanation of a true fact is most likely to be

true (Peirce, 1965), one could assume that all the inferences from the most simple and plausible justification tree are correct.

With this assumption, the KB is improved by:

- learning a new rule by empirical inductive generalization:

$\text{speed}(X, \text{high}) :- \text{processor}(X, \text{risc})$

with the positive examples

$X=\text{sun01}, X=\text{hp05}, X=\text{xerox01}, X=\text{proprinter01}$

- discovering positive examples of the determination rule (which is therefore enforced):

$\text{runs}(X, Y) :- \text{os}(X, Z).$

with the positive examples

$X=\text{macII03}, Y=\text{page-maker}, Z=\text{mac-os}$

$X=\text{macII02}, Y=\text{page-maker}, Z=\text{mac-os}$

- discovering positive examples for the deductive rules used in building the plausible justification tree as, for instance:

$\text{suitable}(X, \text{publishing}) :-$

$\text{runs}(X, \text{publishing-sw}), \text{communicate}(X, Y),$

$\text{isa}(Y, \text{high-quality-printer}).$

with the positive example

$X=\text{macII02}, Y=\text{proprinter01}$

Therefore, the user merely verifying a statement allows the system to refine the KB by making several justified hypotheses. As a result of these improvements

$\text{KB} \models \text{suitable}(\text{macII02}, \text{publishing})$

During KB refinement, the rules are constantly updated so as to remain consistent with the accumulated examples. This is a type of incremental learning with full memory of past examples.

As mentioned before, the input fact "suitable(macII02, publishing)" is representative for the kind of answers the final system

should be able to generate. This means that the final system should be able to give other answers of the form "suitable(x, y)". It is therefore desirable to extend *DC* so as to include other such true facts, but also to improve *DC* so as no longer to include false facts of the same form.

While the integration of the fact "suitable(macII02, publishing)" into *DC* was a costly process that involved multitype inferences and the determination of the most plausible justification tree, the integration in (or exclusion from) *DC* of similar facts is a much simpler process which basically replicates most of the reasoning involved in the "understanding" of the input "suitable(macII02, publishing)". This feature is one of the main strengths of the proposed KB refinement method.

The basic idea is the following one. One performs a costly reasoning process to show that $KB \models I$. Then it computes a generalization of that reasoning so as to speed up future problem solving which requires a similar reasoning. Indeed, such a reasoning process could be generated by simply instantiating this generalization to the new problem to be solved.

A simple illustration of this idea is explanation-based learning (Mitchell, Keller, Kedar-Cabelli, 1986; DeJong and Mooney, 1986). In this case, an explanation (proof tree) of a concept example is deductively generalized. Different instances of this deductive generalization demonstrate that other descriptions are examples of the same concept.

In our method, each inference from the plausible justification tree is replaced with a generalization which depends of the type of inference, as shown in (Tecuci, 1993). Thus, the system is performing not only deductive generalizations, but also empirical inductive generalizations, generalizations based on different types of analogies, and possibly, even generalizations based on abduction. To illustrate this, let us consider the analogical implication from Figure 5. The process of making this inference is illustrated in Figure 6.

According to the plausible determination rule "runs(X, Y) :- os(X, Z)", the software which a machine can run is largely determined by its operating system. It is known that the oper-

ating system of "macIIc03" is "mac-os", and that it runs "page-maker". Because the operating system of "macII02" is also "mac-os", one may infer by analogy that "macII02" could also run "page-maker".

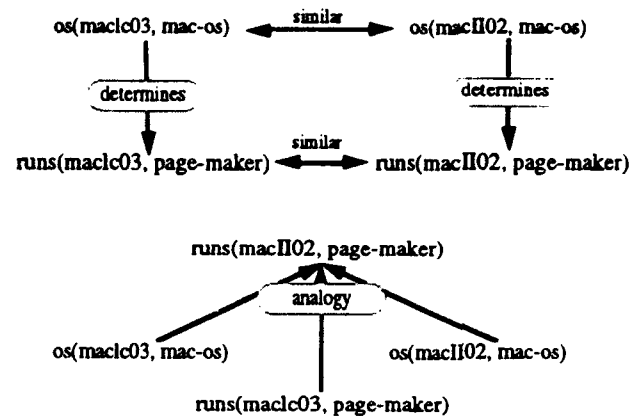


Figure 6: Inferring "runs(macII02, page-maker)" by analogy.

Let us notice now that the same kind of reasoning is valid for any type of operating system, and for any type of software, as illustrated in Figure 7.

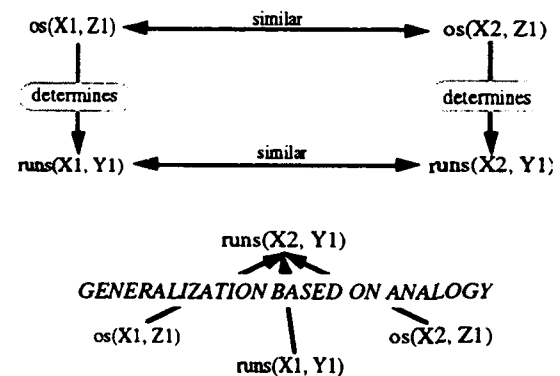


Figure 7: Generalization of the reasoning illustrated in Figure 6.

Now, if one knows, for instance, that "os(hp05, unix)", "runs(hp05, frame-maker)", and "os(sun01, unix)", then one may immediately infer "runs(sun01, frame-maker)", by simply instantiating the general inference from the bottom of Figure 7.

This might not appear to be a significant saving but, in the case of a plausible justification tree, one generalizes several such individual inferences and, even more importantly, their interconnection in a plausible reasoning process. For instance, the

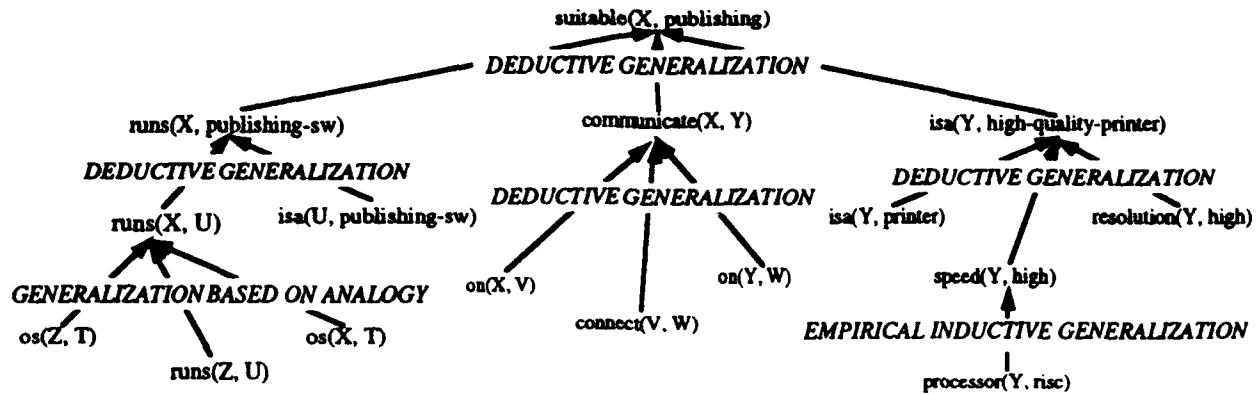


Figure 8: A generalized plausible justification tree.

generalization of the tree in Figure 5 is presented in Figure 8. The generalization method is presented in (Tecuci, 1993).

The important thing about the general tree in Figure 8 is that it covers many of the plausible justification trees for facts of the form "suitable(x, publishing)". If, for instance, sun01 is another computer for which the leaves of the plausible justification tree in Figure 8 are true, then the system will infer that sun01 is also suitable for publishing, by simply instantiating the tree in Figure 8 (see Figure 9).

Let us mention again that while building the plausible justification tree in Figure 5 was a difficult problem which required the employment of different types of reasoning, and of determining the simplest and the most plausible justification tree, the building of the tree in Figure 9 was a very simple process of instantiating the tree in Figure 8. However,

once this tree is built, one may draw conclusions that are similar to the ones drawn from the tree in Figure 5:

- if the top of the tree in Figure 9 is known to be true, then one may assume that all the intermediate implications are also valid. This reinforces each rule used in building the tree with a new positive example.
- if the top of the tree is not true, then one has to identify the wrong implication and to correct accordingly the KB.

6. Experimentation, verification and repair

Building the plausible justification tree from Figure 5 and its generalization from Figure 8 was the first stage of the KB refinement process described in Figure 3. The next stage is one of experimentation, verification, and repair.

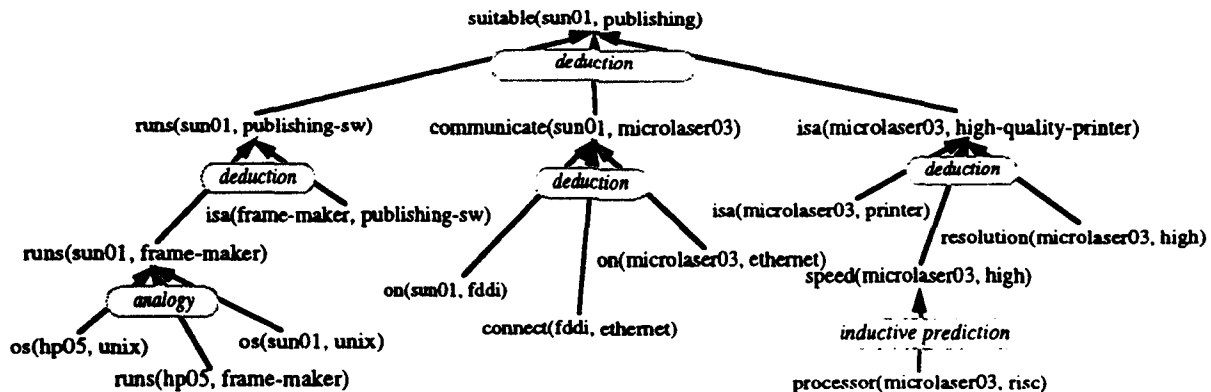


Figure 9: An instance of the plausible justification tree in Figure 8, justifying that sun01 is suitable for publishing.

The system will generate plausible justification trees like the one in Figure 9. These trees show how statements of the form "suitable(x, publishing)" plausibly derive from the KB. Each such statement is shown to the user who is asked if it is true or false. Then, the system (with the expert's help) will update the KB such that it will deductively entail the true statements and only them.

The experimentation phase is controlled by a heuristic search in a plausible version space (PVS) which limits significantly the number of experiments needed to improve the KB. In the case of our example, the plausible version space is defined by the trees in Figure 5 and Figure 8, and is represented in Figure 10.

plausible upper bound
 suitable(X, publishing) :-
 os(Z, T), runs(Z, U), os(X, T),
 isa(U, publishing-sw), on(X, V),
 connect(V, W), on(Y, W),
 isa(Y, printer), processor(Y, risc),
 resolution(Y, high).

plausible lower bound
 suitable(macII02, publishing) :-
 os(macII03, mac-os), runs(macII03, page-maker),
 os(macII02, mac-os),
 isa(page-maker, publishing-sw),
 on(macII02, appletalk), connect(appletalk, ethernet),
 on(proprinter01, ethernet), isa(proprinter01, printer),
 processor(proprinter01, risc),
 resolution(proprinter01, high).

Figure 10: The plausible version space (PVS)

The plausible upper bound is a rule the left hand side of which is the top of the general tree in Figure 8, and the right hand side of which is the conjunction of the leaves of the same tree. The plausible lower bound is a similar rule corresponding to the tree in Figure 5. This plausible version space synthesizes some of the inferential capabilities of the system with respect to the facts of the form "suitable(x, publishing)". We call these bounds plausible because they are only approximations of the real bounds (Tecuci, 1992). The upper bound rule is supposed to be more general than the exact rule for inferring "suitable(x, publishing)", and the lower bound rule is supposed to be less general than this rule. Let us notice that this version space corresponds to the version space in Figure 2. The plausible upper bound

corresponds to the plausible closure, and the plausible lower bound corresponds to the deductive closure. Of course, this space is restricted to facts of the form "suitable(x, publishing)".

The version space in Figure 10 could be represented in the equivalent form in Figure 11. Note that the facts of the form "isa(Q, something)" are always true.

suitable(X, publishing) :-

plausible upper bound
 isa(T, something), isa(U, publishing-sw),
 isa(V, something), isa(W, something),
 isa(X, something), isa(Y, printer),
 isa(Z, something), os(Z, T), runs(Z, U), os(X, T),
 on(X, V), connect(V, W), on(Y, W),
 processor(Y, risc), resolution(Y, high).

plausible lower bound
 isa(T, mac-os), isa(U, publishing-sw),
 isa(V, appletalk), isa(W, ethernet),
 isa(X, macII02), isa(Y, printer),
 isa(Z, macII03), os(Z, T), runs(Z, U), os(X, T),
 on(X, V), connect(V, W), on(Y, W),
 processor(Y, risc), resolution(Y, high).

with the positive example
 T=mac-os, U=page-maker, V=appletalk,
 W=ethernet, X=macII02, Y=proprinter01,
 Z=macII03.

Figure 11: Equivalent form of the plausible version space in Figure 10.

The version space in Figure 11 serves both for generating facts of the form "suitable(x, publishing)", and for determining the end of the experimentation phase.

To generate such a fact, the system looks into the KB for an instance of the upper bound which is not an instance of the lower bound. Such an instance is the following one:

suitable(sun01, publishing) :-
 os(hp05, unix),
 runs(hp05, frame-maker),
 os(sun01, unix),
 isa(frame-maker, publishing-sw),
 on(sun01, fddi),
 connect(fddi, ethernet),
 on(microlaser03, ethernet),
 isa(microlaser03, printer),
 processor(microlaser03, risc),
 resolution(microlaser03, high).

which could be written as:

suitable(X, publishing) :-
 isa(T, unix), isa(U, publishing-sw), isa(V, fddi),
 isa(W, ethernet), isa(X, sun01),
 isa(Y, microlaser03), isa(Z, hp05), os(Z, T),
 runs(Z, U), os(X, T), on(X, V), connect(V, W),
 on(Y, W), processor(Y, risc), resolution(Y, high).

with the positive example

T=unix, U=frame-maker, V=fddi, W=ethernet,
 X=sun01, Y=microlaser03, Z=hp05.

The corresponding instance of the general tree in Figure 8 shows how "suitable(sun01, publishing)" is plausibly entailed by the KB (see Figure 9). The user is asked if "suitable(sun01, publishing)" is true or false, and the KB is updated accordingly. Assuming the user accepted "suitable(sun01, publishing)" as a true fact, the KB and the plausible version space are updated as follows:

- the KB is improved so as to deductively entail "suitable(sun01, publishing)";
- the plausible lower bound of the PVS is conjunctively generalized to "cover" the leaves of the tree in Figure 9.

It has already been shown how the KB is improved (see section 5). The plausible lower bound of the PVS is generalized as shown in Figure 12.

Let us also consider the case of a generated fact which is rejected by the user:

"suitable(macplus07, publishing)".

The corresponding plausible justification tree is shown in Figure 13. This tree was obtained by instantiating the general tree in Figure 8 with facts from the KB. It shows how a false fact is plausibly entailed by the KB.

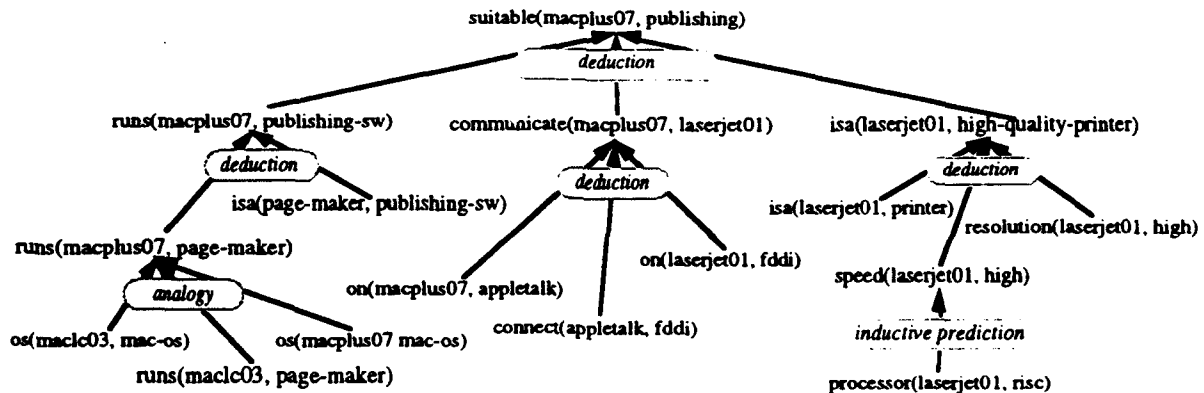


Figure 13: Another instance of the plausible justification tree in Figure 8, which shows how a false fact is plausibly entailed by the KB.

suitable(X, publishing) :-

plausible upper bound

isa(T, something), isa(U, publishing-sw),
 isa(V, something), isa(W, something),
 isa(X, something), isa(Y, printer),
 isa(Z, something), os(Z, T), runs(Z, U), os(X, T),
 on(X, V), connect(V, W), on(Y, W),
 processor(Y, risc), resolution(Y, high).

plausible lower bound

isa(T, op-system), isa(U, publishing-sw),
 isa(V, network), isa(W, ethernet),
 isa(X, workstation), isa(Y, printer),
 isa(Z, workstation), os(Z, T), runs(Z, U), os(X, T),
 on(X, V), connect(V, W), on(Y, W),
 processor(Y, risc), resolution(Y, high).

with the positive example

T=mac-os, U=page-maker, V=appletalk, W=ethernet,
 X=macII02, Y=proprinter01, Z=macII03.
 T=unix, U=frame-maker, V=fddi, W=ethernet,
 X=sun01, Y=microlaser03, Z=hp05.

Figure 12: Updated PVS.

In such a case one has to detect and correct the wrong inference(s), as well as to update the KB, the general justification tree in Figure 8, and the plausible version space such that:

- the tree in Figure 13 is no longer a plausible justification tree;
- the KB does not deductively entail "suitable(macplus07, publishing)";
- the updated general justification tree no longer covers the tree in Figure 13;
- the plausible upper bound of the PVS is specialized so that it no longer covers the leaves of the tree in Figure 13.

Detecting the wrong implication from the plausible justification tree in Figure 13 is an intrinsically difficult problem for an autonomous learning system. One possible solution, which is presented in (Tecuci, 1993), is to blame the implication which is the least plausible, and the correction of which requires the smallest change in the KB. For a human expert, however, it should not be too difficult to identify the wrong implication and even to find the explanation of the failure (Tecuci, 1992). In the case of the tree in Figure 13, the wrong implication could be identified by the user as being the deduction from the top of the tree:

```
suitable(macplus07, publishing) :-
    runs(macplus07, publishing-sw),
    communicate(macplus07, laserjet01),
    isa(laserjet01, high-quality-printer).
```

Although macplus07 runs publishing software and communicates with a high quality printer, it is not suitable for publishing because it does not have a large display.

Consequently, the rule which generated the above implication is specialized as follows (requiring X to have a large display):

```
suitable(X, publishing) :-
    runs(X, publishing-sw), display(X, large),
    communicate(X, Y), isa(Y, high-quality-printer).
with the positive examples
    X=macII02, Y=proprinter01.
    X=sun01, Y=microlaser03.
with the negative example
    X=macplus07, Y=laserjet01.
```

One should notice that the predicate "display(X, Y)" could be defined by the user, or could be suggested by the system as one which distinguishes the known positive exam-

ples of the rule from the discovered negative example.

As a result of updating the above rule, the general plausible justification tree in Figure 8 is updated as shown in Figure 14, and the version space is updated as shown in Figure 15.

It might not always be easy to identify the problem with a wrong inference, and to specialize the corresponding rule so as no longer to cover the negative example. In such a case, the wrong inference is kept as a negative exception of the rule which generated it, as shown in Figure 16.

```
suitable(X, publishing) :-
    plausible upper bound
    isa(T, something), isa(U, publishing-sw),
    isa(V, something), isa(W, something),
    isa(X, something), isa(Y, printer), isa(Z, something),
    os(Z, T), runs(Z, U), os(X, T), display(X, large),
    on(X, V), connect(V, W), on(Y, W),
    processor(Y, risc), resolution(Y, high).
    plausible lower bound
    isa(T, op-system), isa(U, publishing-sw),
    isa(V, network), isa(W, ethernet),
    isa(X, workstation), isa(Y, printer),
    isa(Z, workstation), os(Z, T), runs(Z, U), os(X, T),
    display(X, large), on(X, V), connect(V, W),
    on(Y, W), processor(Y, risc), resolution(Y, high).
with the positive example
    T=mac-os, U=page-maker, V=appletalk, W=ethernet,
    X=macII02, Y=proprinter01, Z=macIc03.
    T=unix, U=frame-maker, V=fddi, W=ethernet,
    X=sun01, Y=microlaser03, Z=hp05.
with the negative example
    T=mac-os, U=page-maker, V=appletalk, W=fddi,
    X=macplus07, Y=laserjet01, Z=macIc03.
```

Figure 15: Updated PVS.

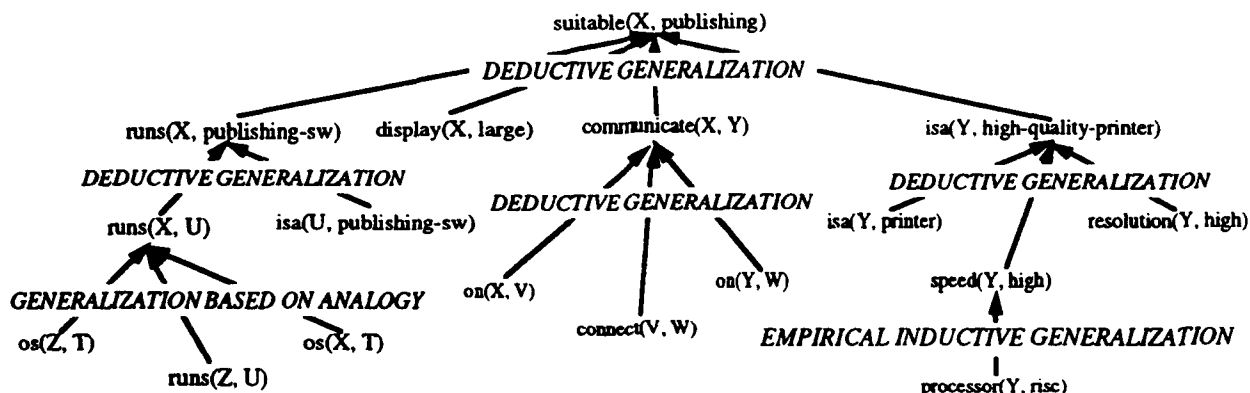


Figure 14: Updated general justification tree.

```

suitable(X, publishing) :-
  runs(X, publishing-sw), communicate(X, Y),
  isa(Y, high-quality-printer).
  with the positive examples
    X=macII02, Y=proprinter01.
    X=sun01, Y=microlaser03.
  with the negative exception
    X=macplus07, Y=laserjet01.

```

Figure 16: A rule with a negative exception.

During experimentation, the lower bound of the plausible version space is generalized so as to cover the generated facts accepted by the user (the positive examples), and the upper and lower bounds are specialized so as to no longer cover the generated facts rejected by the user (the negative examples). This process will end in one of the following situations:

- the bounds of the plausible version space become identical.
- the bounds are not identical, but the KB no longer contains any instance of the upper bound of the version space that is not an instance of the lower bound. Therefore, no new fact of the form "suitable(x, publishing)" can be generated.

Notice that the plausible version space is only used for controlling the experimentation phase. It is not kept in the KB as a new rule for inferring "suitable(x, publishing)" because it would be a redundant rule.

7. Goal-driven knowledge elicitation

Because the KB is incomplete and partially incorrect, some of the learned knowledge pieces may be inconsistent (i.e. may cover negative examples), as it is illustrated in Figure 16. In order to remove such inconsistencies, additional knowledge pieces (which may represent new terms in the representation language of the system) are elicited from the expert, through several consistency-driven knowledge elicitation methods, as described in (Tecuci and Hieb, 1993). These methods are applied in the third phase of KB refinement, as shown in Figure 3.

Let us consider the case of the inconsistent rule in Figure 16.

One consistency-driven knowledge elicitation method is to look for a new predicate which could characterize all the positive instances of

X (although it might not be associated with each of these instances), without characterizing any negative exception of X. A potentially discriminating predicate like "display" is one which characterizes a positive instance of X (either macII02 or sun01), and does not characterize the negative exception of X (macplus07). If such a predicate is found, then the user is asked if it characterizes all the other positive instances of X. The same technique could, of course, be applied to the instances of Y.

It may happen, however, that the system cannot find a property to transfer from one positive example of X to the others. In such a case, it will try to elicit a new property by using a technique similar to the triad method employed in the elicitation of the repertory grids (Boose and Bradshaw, 1988; Shaw and Gaines, 1987).

Another method for removing the negative exception is to look for a relationship between X and Y which could characterize all the positive instances of X and Y, without characterizing the negative exception.

Yet another method is to define a new concept that covers all the positive instances of X (or all the positive instances of Y), without covering the negative exception of X (Y). A method similar to this one is reported by (Wrobel, 1989).

8. Summary and conclusions

Figure 2 suggests two basic approaches to the development of the competence of a deductive knowledge-based system. One approach is to extend the deductive closure of the KB by acquiring new knowledge. The other approach is to replace the deductive inference engine with a plausible inference engine, and thus to enable the system to solve additional problems from the plausible closure. The first approach has the advantage that the system employs "sound" reasoning, but it has the disadvantage of requiring a difficult knowledge acquisition process. The second approach has the advantage of avoiding knowledge acquisition, but the disadvantage that the system needs to rely on plausible reasoning.

The knowledge refinement method presented in this paper is an attempt to combine these

two approaches in such a way as to take advantage of their complementarity. This method is summarized in Figure 17. It resulted from the merging and extension of two related research directions:

- the knowledge acquisition methodology of Disciple (Tecuci and Kodratoff, 1990) and NeoDisciple (Tecuci, 1992);
- the MTL-JT framework for multistrategy learning based on plausible justification trees (Tecuci, 1993).

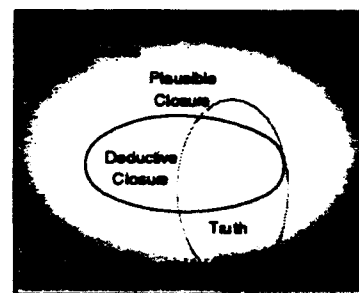
On the one hand, it extends NeoDisciple with respect to the knowledge representation used and the types of inferences and generalizations employed and, on the other hand, it adapts and integrates the MTL-JT framework into an interactive knowledge acquisition scenario.

The method is based on the following general idea. The system performs a complex reasoning process to solve some problem P . Then it determines a justified generalization of the reasoning process so as to speed up the process of solving similar problems P_i . When the system encounters such a similar problem, it will be able to find a solution just by instantiating the above generalization.

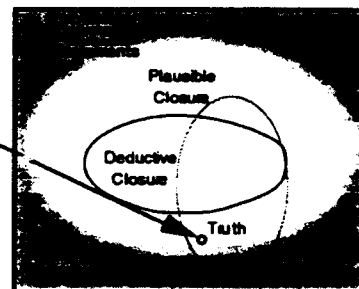
In the context of the presented method, the problem to solve is to extend the KB so as to entail a new fact I . The complex reasoning process involved consists of building a plausible justification tree. This reasoning process is generalized by employing various types of generalization procedures. Then, during the experimentation phase, the system instantiates this generalization and, using it, improves the KB so as to entail similar facts which are true (or to no longer entail similar facts which are false).

One important aspect of the presented method is the notion of plausible justification trees (Tecuci and Michalski, 1991; Tecuci, 1993). Other systems have employed implicit justification trees (DeRaedt and Bruynooghe, 1993), or even explicit justification trees (Tecuci, 1988; Mahadevan, 1989; Widmer, 1989), that integrated only a small number of inferences. In our method, the plausible justification tree is defined as a general framework for integrating a whole range of inference types. Therefore, theoretically, there is no limit with respect to the type or

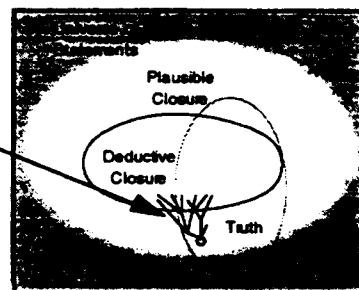
Initial state, showing Truth and the Plausible and Deductive Closures.



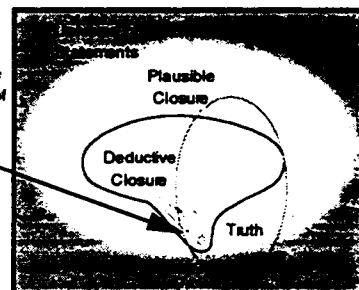
Here, a new input fact is introduced. The fact is outside the initial deductive closure.



A plausible justification tree for the new fact is constructed via multitype inference. This justification connects the new fact with statements in the original deductive and plausible closures.



A generalization of the plausible justification tree becomes part of the deductive closure. DC has now been extended to cover more of TC.



Through experimentation and consistency-driven knowledge elicitation, DC may be further refined reducing the sets DC-TC and TC-DC. +'s indicate positive instances, -'s indicate negative instances.

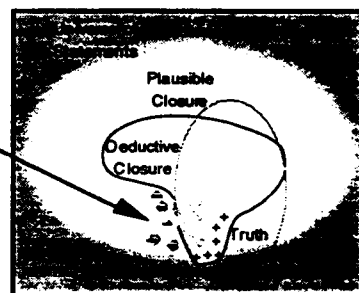


Figure 17: Modification of DC and PC during KB refinement.

number of inferences employed in a plausible justification tree.

Another important feature of the KB refinement method is the employment of

different types of generalizations. While the current machine learning research distinguishes only between deductive generalizations and inductive generalizations (Michalski, 1993), this method and the MTL-JT framework (Tecuci, 1993) suggest that one may consider other types of generalizations, each associated with a certain type of inference (as, for instance, generalization based on analogy).

There are also several ways in which the method could be improved. For instance, the set of inferences involved in the present version of the method is quite limited (deduction, determination-based analogy, inductive prediction, and abduction). New types of inferences should be included, as well as more complex versions of the current ones.

Also, new types of justified generalizations (each corresponding to a certain inference type) should be defined.

Finally, the goal-driven knowledge elicitation methods briefly mentioned in section 7 should be extended so as not only to add new concepts and relationships into the KB but also to delete those that become unnecessary.

Acknowledgments

The authors are grateful to Luc De Raedt and Michael Hieb for useful comments and criticisms. This research was done in the GMU Center for Artificial Intelligence. Research of the Center is supported in part by the National Science Foundation Grant No. IRI-9020266, in part by the Office of Naval Research grant No. N0014-91-J-1351, and in part by the Advanced Research Projects Agency grant No. N0014-91-J-1854, administered by the Office of Naval Research, and grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research.

References

- Blumer A., Ehrenfeucht A., Haussler D. and Warmuth M. K., Occam's Razor, *Information Processing Letters*, 24, pp.377-380, 1987.
- Boose J.H. and Bradshaw J.M., Expertise Transfer and Complex Problems: Using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems, in *Knowledge Acquisition Tools for Expert Systems*, J.Boose and B.Gaines (Eds.), AP, 1988.
- Cain T., DUCTOR: A Theory Revision System for Propositional Domains, in *Machine Learning: Proc. of the Eighth Int. Workshop*, L.A. Birnbaum and G.C. Collins (Eds.), Chicago, IL, Morgan Kaufmann, 1991.
- Carbonell J.G., Derivational Analogy: a Theory of Reconstructive Problem Solving and Expertise Acquisition, in *Machine Learning: An Artificial Intelligence Approach*, (Vol. 2), R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds), San Mateo, CA: Morgan Kaufmann, 1986.
- Cohen W., The Generality of Overgenerality, in *Machine Learning: Proc. of the Eighth Int. Workshop*, L.A. Birnbaum and G.C. Collins (Eds.), Chicago, Morgan Kaufmann, 1991.
- Collins A., and Michalski R.S., The Logic of Plausible Reasoning: A Core Theory, *Cognitive Science*, Vol. 13, pp. 1-49, 1989.
- Davies T.R. and Russell S.J., A Logical Approach to Reasoning by Analogy, *Proc. IJCAI*, Milan, Italy, Morgan Kaufmann, 1987.
- DeJong G. and Mooney R., Explanation-Based Learning: An Alternative View, *Machine Learning*, Vol. 1, 1986.
- Ginsberg A., Weiss S.M. and Politakis P., Automatic Knowledge Base Refinement for Classification Systems, *Artificial Intelligence* 35, pp. 197-226, 1988.
- Gentner D., The Mechanisms of Analogical Reasoning, in *Readings in Machine Learning*, J.W.Shavlik, T.G.Dietterich (Eds), Morgan Kaufmann, 1990.
- Greiner R., Learning by Understanding Analogies, in *Analogica*, A. Prieditis (Ed.), Pitman, London, 1988.
- Josephson J., Abduction: Conceptual Analysis of a Fundamental Pattern of Inference, Research Report 91-JJ, Laboratory for AI Research, Ohio State University, 1991.
- Kedar-Cabelli S., Toward a Computational Model of Purpose-Directed Analogy, in *Analogica*, A. Prieditis (Ed.), Pitman, 1988.
- Kodratoff Y., Using Abductive Recovery of Failed Proofs for Problem Solving by Analogy, in *Machine Learning: Proc. of the Seventh Int. Workshop*, B.W. Porter and R.J.

Mooney (Eds.), Morgan Kaufmann, 1990.

Lee O., A Control Strategy for Multistrategy Task-adaptive Learning with Plausible Justification Trees, Research Report, AI Center, George Mason University, 1993.

Mahadevan S., Using Determinations in Explanation-based Learning: A Solution to Incomplete Theory Problem, in *Proc. of the Sixth Int. Workshop on Machine Learning*, Ithaca, NY: Morgan Kaufman, 1989.

Michalski R.S., A Theory and Methodology of Inductive Learning, in *Machine Learning: An Artificial Intelligence Approach* (Vol. 1) R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga Publishing Co., 1983.

Michalski R.S., Inferential Learning Theory: Developing Theoretical Foundations for Multistrategy Learning, in *Machine Learning: A Multistrategy Approach*, Vol. 4, R.S. Michalski and G. Tecuci (Eds.), San Mateo, CA, Morgan Kaufmann, 1993.

Mitchell T.M., Version Spaces: An Approach to Concept Learning, Doctoral dissertation, Stanford University, 1978.

Mitchell T.M., Keller T., and Kedar-Cabelli S., Explanation-Based Generalization: A Unifying View, *Machine Learning*, Vol. 1, pp. 47-80, 1986.

Mooney R.J. and Ourston D., A Multistrategy Approach to Theory Refinement, in *Machine Learning: An Artificial Intelligence Approach*, Vol. 4, R.S. Michalski and G. Tecuci (Eds.), San Mateo, CA, Morgan Kaufmann, 1993.

Pople H.E., On the Mechanization of Abductive Logic, *Proc. IJCAI*, Stanford, CA, William Kaufmann, Inc., pp. 147-152, 1973.

Peirce C. S., Elements of Logic, in *Collected Papers of Charles Sanders Peirce (1839-1914)*, Ch. Hartshorne and P. Weiss (Eds.), The Belknap Press Harvard University Press, Cambridge, MA, 1965.

Quinlan J. R., Induction of Decision Trees, *Machine Learning* 1, pp. 81-106, 1986.

De Raedt L., Interactive Concept Learning, Ph.D. Thesis, Catholic Univ. Leuven, 1991.

De Raedt L. and Bruynooghe M., Interactive Theory Revision, in *Machine Learning: A Multistrategy Approach*, Vol. 4, R.S. Michalski and G. Tecuci (Eds.), San Mateo, CA, Morgan Kaufmann, 1993.

Richards B.L. and Mooney R.J., First-Order Theory Revision, to appear, 1993.

Russell S.J., *The Use of Knowledge in Analogy and Induction*, Pitman, 1989.

Shaw M.L.G. and Gaines B.R., An interactive knowledge elicitation technique using personal construct technology, in *Knowledge Acquisition for Expert Systems: A Practical Handbook*, A.L. Kidd (Ed.), Plenum Press, 1987.

Tecuci G., DISCIPLE: A Theory, Methodology, and System for Learning Expert Knowledge, Ph.D. Thesis, University of Paris-South, 1988.

Tecuci G., Automating Knowledge Acquisition as Extending, Updating, and Improving a Knowledge Base, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 22, No. 6, 1992.

Tecuci G., Plausible Justification Trees: a Framework for the Deep and Dynamic Integration of Learning Strategies, *Machine Learning*, Vol.4&5, 1993.

Tecuci G. and Hieb M., Consistency-driven Knowledge Elicitation: Using a Machine Learning Oriented Knowledge Representation to Integrate Learning and Knowledge Elicitation in NeoDISCIPLE, *Knowledge Acquisition Journal*, to appear 1993.

Tecuci G. and Kodratoff Y., Apprenticeship Learning in Imperfect Theory Domains, in *Machine Learning: An Artificial Intelligence Approach*, Vol. 3, Y. Kodratoff and R.S. Michalski (Eds.), San Mateo, CA, Morgan Kaufmann, 1990.

Tecuci G. and Michalski R.S., A Method for Multistrategy Task-adaptive Learning Based on Plausible Justifications, in L. Birnbaum and G. Collins (eds), *Machine Learning: Proc. of the Eighth International Workshop*, Chicago, IL: Morgan Kaufmann, 1991.

Widmer, G., A Tight Integration of Deductive and Inductive Learning, in *Proc. of the Sixth Int. Workshop on Machine Learning*, Ithaca, NY: Morgan Kaufman, 1989.

Winston P.H., Learning and Reasoning by Analogy, *Communications of the ACM*, Vol. 23, 1980.

Wrobel S., Demand-driven Concept Formation, in Morik K. (Ed.), *Knowledge Representation and Organization in Machine Learning*, Springer Verlag, 1989.

Learning to Survive

Luc De Raedt, Erlend Bleken, Vincent Coget,
 Chaouat Ghil, Bart Swennen and Maurice Bruynooghe
 Department of Computing Science, Katholieke Universiteit Leuven
 Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Abstract

A novel learning task, that of learning to survive in an unknown and hostile environment, is defined and explored. The environment is incorporated in an adventure-game of the "hack" family and shares several important features with the real world: the actions are non-deterministic, the agents possess only partial knowledge about their performance and about the world, and they have only a limited influence on their environment. Central in the adventure game is the autonomous agent, which consists of a planning and a learning component. This agent (and its opponents) can execute actions, which will change the environment; some actions will have desirable effects with respect to the agent's goals, others will not. Initially, the effects of the actions and the behavior of the opponents is unknown to the agent. In order to survive in the hostile environment, the agent has to learn the effects of the actions, the behaviour of its opponents, to evaluate the situation it is in, and to execute the appropriate actions. To this aim, we have implemented an agent incorporating multiple strategies for learning: learning by experientia-

tion (knowledge discovery), empirical induction on examples (using inductive logic programming), learning control knowledge (by modifying evaluation functions), and learning from experience (using a knowledge base manager).

1 Introduction












Central to intelligence (whether natural or artificial) is the ability to adapt to and to perform well in an unknown environment. Whereas the real world is still a very complex environment for artificial intelligence, some fundamental properties of the real world can easily be modelled in artificial worlds such as adventure games. Although such artificial worlds are usually considered as toy-domains, they often share important properties with the real world, which means that understanding intelligence in artificial worlds can enhance our general understanding of intelligence. At the same time, artificial worlds have the advantage that their complexity can easily be controlled, making feasible a stepwise introduction of real world characteristics.

In this paper, we explore an artificial world of an adventure game of the "hack" or "rogue" family [Raymond and Threepoint, ; Stephenson,]. In this type of game, the player controls an agent living in a hostile environment randomly generated on the board. The environ-

ment is hostile in the sense that other agents live in the same environment and may attack the player's agent. To survive, the player has to collect items (such as food, weapons, etc.), to eat and drink at regular times and to kill his opponents. This kind of game has the following real world characteristics: agents possess only partial knowledge of their current situation and their performance, to survive they have to learn the effects of their actions and the behavior of their opponents, and they have to use that knowledge in order to select appropriate actions to execute. The main contribution of this work is the design and implementation of an intelligent agent, which learns to survive and to improve its performance in a non-trivial artificial world. Furthermore, because the agent's most distinct characteristic is its ability to learn using multiple strategies (and it seems very hard to survive using a single state-of-the-art learning strategy, if not impossible), our work provides evidence that multistrategy learning is central to intelligence. Our agent learns different types of knowledge: rules to predict the effects of actions and the behavior of opponents (using empirical learning from examples), an evaluation function to assess the degree of performance (learning control knowledge), and our agent maintains only the interesting rules in its knowledge base (learning from experience using a knowledge base manager). It uses a kind of minimax algorithm as planner to select the next action to execute.

This paper is organized as follows: in Section 2, we discuss the artificial world incorporated in the adventure game, in Section 3, we present the overall architecture of the learning system, in Section 4, we discuss rule generation and knowledge base management, in Section 5, we show how the evaluation function is learned, in Section 6, we present the planner, in Section 7, we report on the current state of the system and some experiments, finally in Section 8, we conclude and touch briefly on related work.

2 The environment

The game is played using the graphical interface shown in Figure 1. The available knowledge about the current situation of the learning agent (or the human player, if the game is played by humans) is shown on a board of 9 positions (the board shown in Figure 1 contains 49 positions, but only the 9 central positions are observable by the player). The position in the middle of the board is always occupied by the learning agent. The position of the learning agent is identified by the number 0 and the agent is represented by . Neighbouring positions have numbers ranging from 1 to 8. The neighbouring positions may contain opponents (Humans , Dragons  and Bats ) and objects (Gold , Food , Potions , Armors , Handweapons , Wands , and spells  of different types); see the Appendix for more details. For example, in Figure 1, the agent is next to a Bat (B) on position 5, next to food (on position 2, next to a wall (*), and next to a position containing several objects (#). Objects can be cursed; cursed objects behave differently than uncursed ones. Curses can be added or removed by quaffing potions, casting spells or zapping wands. Objects are possessed by agents (the learning agent has an inventory of the objects it owns) or positioned on the board. To survive in the game, the agent has to gather objects and use them as a protection against its opponents. The aim of the game is twofold: to become rich (i.e. acquire as much gold as possible) and to maximize the energy level. The energy level is the only means of direct performance evaluation. It decreases when the agent is hit by opponents, when it hits one of its opponents or when it eats cursed food; it increases when the agent consumes uncursed food. If the energy level falls below zero, the agent dies.

The game operates in a cyclic process. In the first step of each cycle, the learning agent can execute an action and in the second step of

each cycle, all its opponents may execute an action. A list of possible actions is specified in Appendix 1. The cyclic process continues until the agent dies or the user stops the game.

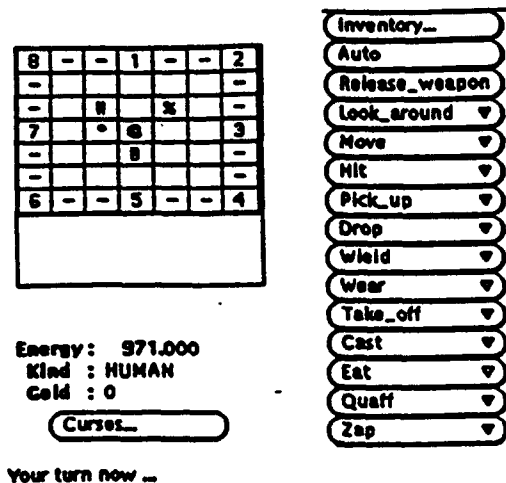


Figure 1 : graphical interface of the game.

Important for our scientific purposes is that the actions are non deterministic, that the only direct evaluation criteria our agent has access to is the energy level and the number of gold pieces, and that when the agent moves towards previously unvisited positions, they are filled out randomly. Non deterministic in our context means that executing the same action twice in the same situation may yield different effects (determined by a random generator). For instance, when hitting an agent, three effects can occur: the agent misses, the agent hits the other agent (and the other agent's energy level decreases), the agent hits the other agent (and the other agent's energy level falls below zero, resulting in the death of the other agent).

The initial knowledge of the game our agent starts from is similar to that of a novice human player. More specifically, the ONLY initial knowledge our agent possesses, is the fol-

lowing:

- complete knowledge of the effects of the move action;
- for all other actions, a list of determinations [Russell, 1989] specifying for each action, the possibly relevant literals; determinations capture the intuitive knowledge a novice human player has about the actions that can be executed; for instance, for the actions drop and pick up only objects possessed or on the current position of the agent and general characteristics (such as curses, energy and gold) are relevant whereas the opponents and the neighbouring positions are irrelevant; for the actions wear and take off only features related to armors and general characteristics are relevant; the determinations are used to remove irrelevant literals when constructing examples, i.e. only literals matching the determinations are included in the example descriptions;
- a list of general features influencing the aim of the game (control knowledge); this knowledge is used by the evaluation function learner (see Section 6) and is basically a list of features such as the number of armors, the number of weapons, the weapons being wield, the number of opponents on neighbouring positions, etc.

We believe such knowledge corresponds to prior assumptions and expectations each rational agent has (and should have) when operating in this domain. Learning without such knowledge is possible as well, but is much slower and slightly harder.

3 The Autonomous Agent's Architecture

Figure 2 shows the architecture of the system, its main components and the information flow between the different components.

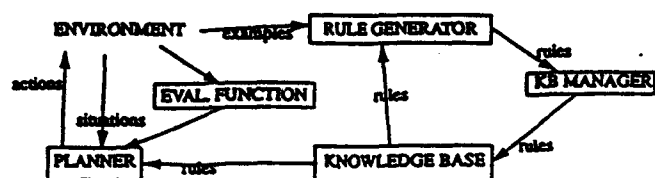


Figure 2: The Agent's Architecture

In the two steps of the game cycle, the current situation of the game is transformed in the next one. In the first step, this is done by executing the action selected by the autonomous agent, whereas the transformation of the second step is determined by the game. Each transformation is encoded in a symbolic description, which is passed to the rule generator as an example. The rule generator possibly generalizes (using inductive logic programming) the rules in the knowledge base with the given example, and passes the resulting generalizations (or the example when no good generalization is found) to the knowledge base manager, which may reorganize the knowledge base to accommodate new rules. The task of the knowledge base manager is to decide which rules to keep and which ones to forget. The planner uses the knowledge base and the evaluation function to select the next action to execute. The planner performs a variant of mini max search. The evaluation function is learned by an independent control learning module.

The rules, their generator and the knowledge base manager are described in Section 4; the evaluation function learner in Section 5; and the planner in Section 7. Because the current system is very complex (it is implemented in Prolog by BIM, containing more than 10000 lines (comments not included) of Prolog code), we shall make some slight simplifications in the presentation of the system and focus on concepts rather than on implementation details.

4 Learning and managing rules

4.1 Representing rules and examples

We first introduce some concepts, which are illustrated in Example 1. A *situation description* (of a state in the game) is a conjunction of ground atoms (as perceived by the learning agent). Negated atoms are not explicitly listed in situation descriptions as the *closed world assumption* is being used [Reiter, 1978]. The notation for the closure of a situation description S is $cwa(S)$. Also, rather than applying the closed world assumption as it is, we explicitly introduced negated literals for selected predicates. The list of these predicates and their closure is given in Appendix 2.

For each step in all cycles of the game, an example is constructed. An *example* is composed of four parts: the description S of the situation before the cycle, an atom Act denoting the action executed, and an add- and a remove-list (Al, Rl). The remove list contains the set of a literals that have to be removed from the situation description after executing the action; the add list contains the list of literals to be added to the situation description. This STRIPS-like representation was chosen because of its simplicity and power [Fikes and Nilsson, 1971].

The example generator constructs for each pair of situation descriptions S_1 and S_2 and connecting action A , an example (S, Act, Al, Rl) where $S = S_1 \cup \{\neg l \mid l \in S_2 - S_1\}$ (the second set contains the negation of all literals present after applying the action but not present before); $A = Act$; $Al = cwa(S_2) - cwa(S_1)$; and $Rl = cwa(S_1) - cwa(S_2)$ ¹. Negated literals are only used to

¹Furthermore, the energy and gold level of the situation descriptions S_1 and S_2 are compared. For non-identical levels, the comparison-predicate `less_than` is applied and a literal for this predicate is included in the add-list. Also, literals considered irrelevant for the action are filtered from the situation description (cf. Section 2).

model differences between the two situations.

The structure of rules is a generalization of that of examples. A rule is composed of five parts: a conjunction of literals, denoting the condition part S , an atom Act denoting the action executed, an add- and a remove-list (Al, Rl), and a list of counters C storing information about the correctness, age etc. of the rule. For the moment, we ignore the counters. We shall discuss them in detail in Section 4.3.

A rule $(S_r, Act_r, Al_r, Rl_r, C_r)$ matches a rule $(S_t, Act_t, Al_t, Rl_t, C_t)$ (or an example) if and only if there is a substitution θ such that $S_r\theta \subset S_t$, $Act_r\theta = Act_t$ and $Al_r\theta = Al_t$ and $Rl_r\theta = Rl_t$.

Rules (S, Act, Al, Rl, C) can be used for predicting the resulting situation description S_r when executing action A in situation S_o provided that there is a substitution θ such that $S\theta \subset S_o$ and $Act\theta = A$. The predicted situation description $S_r = S_o - Rl\theta \cup Al\theta$. Rules for the learning agent and the opponent are represented using the same formalism. Actions executed by an opponent, have an extra argument, the agent executing the action.

Two rules $(S_r, Act_r, Al_r, Rl_r, C_r)$ and $(S_t, Act_t, Al_t, Rl_t, C_t)$ are similar if and only if Act_r and Act_t are similar, and (Al_r and Al_t) and (Rl_r and Rl_t) are similar. Two literals are similar if and only if they have the same predicate symbol and sign. Two lists of literals are similar if and only if each literal of the first list is similar to a literal of the second one, and vice versa.

Example 1 : Situations, examples, and rules

A situation description is, for instance:

```
agent_on(1,d), item_on(2,s), obstacle_on(3),
possessing(w), kind_of_agent(d,dragon),
kind(s,spell), kind(w,weapon),
```

```
kind_of_weapon(w,handweapon),
kind_of_hand_weapon(w,dagger),
my_energy(500), my_gold(0), alive, uncursed,
my_kind(human).
```

When executing the action $wield(w)$, the resulting situation description would be:

```
wielding(w), agent_on(1,d), item_on(2,s),
obstacle_on(3), possessing(w),
kind_of_agent(d,dragon),
kind(s,spell), kind(w,weapon),
kind_of_weapon(w,handweapon),
kind_of_hand_weapon(w,dagger),
my_energy(500), my_gold(0), alive, uncursed,
my_kind(human).
```

The example constructed from this transformation would then be (under the assumption that the irrelevant literals for $wield$ are the ones containing information about the neighbouring positions of the agent):

```
S = not_wielding(w), possessing(w),
kind(w,weapon),
kind_of_weapon(w,handweapon),
kind_of_hand_weapon(w,dagger),
my_energy(500), my_gold(0), alive, uncursed,
my_kind(human);
```

```
Act = wield(w);
Al = wielding(w);
Rl = not_wielding(w).
```

Applying this rule to the original situation description indeed results in the transformed situation description. Real-life situation descriptions of game states are usually more complicated as more objects and opponents are involved. \diamond

4.2 Rule generation

Rules in the knowledge base are stored into different classes of similar rules. Each class of rules is organized in a binary tree, where there

is a connection between a parent rule and two children if the parent is the generalization of the children. The most general rule of each class can thus be found in the top of the corresponding tree.

The knowledge base is modified for each incoming example. If there is no rule (class) in the knowledge base similar to the example, a new class of rules containing only the example is added to the knowledge base. If there is a class of rules similar to the example, the corresponding tree is searched for a rule matching the example and all counters of rules matching the example are updated (see below). If no rule matching the example exists, the generalization of the top rule in the tree and the example is computed and added to the tree. The generalization has thus two children: the top rule of the original binary tree and the example. Furthermore the tree may be reorganized according to the knowledge base management principles outlined in the Section 4.3.

The generalization of a rule ($S_r, Act_r, Al_r, Rl_r, C_r$) and an example (S_e, Act_e, Al_e, Rl_e) is based on Plotkin's well-known *lgg*-operator (see [Plotkin, 1970]): The generalized rule ($S_g, Act_g, Al_g, Rl_g, C_g$) is computed as follows (the actual implementation of this algorithm is described in detail in [Bleken, 1992]):

- compute $S_g \rightarrow Act_g = \text{Plotkin's lgg}(S_r \rightarrow Act_r, S_e \rightarrow Act_e)$; this defines S_g and Act_g ;
- determine θ_r and θ_e such that 1) $(S_g \rightarrow Act_g)\theta_r = S_r \rightarrow Act_r$ and 2) $(S_g \rightarrow Act_g)\theta_e = S_e \rightarrow Act_e$;
- determine θ_r^{-1} and θ_e^{-1} such that 3) $Rl_r\theta_r^{-1} \subset S_g$ and 4) $Rl_e\theta_e^{-1} \subset S_g$ and 5) $Rl_r\theta_r^{-1} = Rl_e\theta_e^{-1}$; conditions 3) to 5) are needed for constructing meaningful generalizations because inverse substitutions are not necessarily unique;
- $Al_g := \text{lgg}(Al_r\theta_r^{-1}, Al_e\theta_e^{-1})$ and $Rl_g := Rl_e\theta_e^{-1}$;

- reduce S_g as much as possible without deleting literals from Rl_g using Plotkin's reduction algorithm.

Because the generalization algorithm is quite complex and relies on some concepts from inductive logic programming such as inverse substitutions [Muggleton and Buntine, 1988] and least general generalization [Plotkin, 1970], we illustrate it by Example 2.

Example 2 : Generalizing rules

Consider the following (simplified) example e and rule r . The example starts from a state where there is a bat on position 5 and a human on position 7. When hitting the bat (with bare hands), the bat dies (disappears) and the bat's weapon (an axe) is found on position 5. The meaning of the rule is similar.

$S_e = \text{agent_on}(5,b), \text{kind}(b,\text{Bat}), \text{agent_on}(7,h),$
 $\text{kind}(h,\text{Human}), \text{not_item_on}(5,a);$
 $S_r = \text{agent_on}(7,d), \text{kind}(d,\text{Dragon}),$
 $\text{not_item_on}(7,s);$
 $Act_e = \text{hit}(5);$
 $Act_r = \text{hit}(7);$
 $Rl_e = \text{agent_on}(5,b), \text{kind}(b,\text{Bat}),$
 $\text{not_item_on}(5,a);$
 $Rl_r = \text{agent_on}(7,d), \text{kind}(d,\text{Dragon}),$
 $\text{not_item_on}(7,s);$
 $Al_e = \text{kind}(a,\text{Axe}), \text{item_on}(5,a);$
 $Al_r = \text{kind}(s,\text{Sword}), \text{item_on}(7,s);$

This results in generalizing the following literals together (following Plotkin):

$\text{lgg}(\text{agent_on}(5,b), \text{agent_on}(7,d)) =$
 $\text{agent_on}(P,A)$
 $\text{lgg}(\text{kind}(b,\text{Bat}), \text{kind}(d,\text{Dragon})) = \text{kind}(A,K1)$
 $\text{lgg}(\text{not_item_on}(5,a), \text{not_item_on}(7,s)) =$
 $\text{not_item_on}(P,l)$
 $\text{lgg}(\text{agent_on}(7,h), \text{agent_on}(7,d)) =$
 $\text{agent_on}(7,A2)$
 $\text{lgg}(\text{kind}(h,\text{Human}), \text{kind}(d,\text{Dragon})) =$
 $\text{kind}(A2,K2)$

$$\lgg(\text{hit}(5), \text{hit}(7)) = \text{hit}(P)$$

yielding:

$$\begin{aligned} S_g(\text{non-reduced}) &= \text{agent_on}(P, A), \text{kind}(A, K1), \\ &\text{not_item_on}(P, I), \text{agent_on}(7, A2), \text{kind}(A2, K2) \\ Act_g &= \text{hit}(P); \\ \theta_e &= \{ P = 5, A = b, K1 = \text{Bat}, I = a, A2 = h, K2 = \text{Human} \} \\ \theta_r &= \{ P = 7, A = d, K1 = \text{Dragon}, I = s, A2 = d, K2 = \text{Dragon} \} \end{aligned}$$

θ_e^{-1} is unique (no term appears twice on the right hand side of equations); the inverse substitution for θ_r^{-1} satisfying the requirements is: $\{ 7 \rightarrow P, d \rightarrow A, \text{Dragon} \rightarrow K1, s \rightarrow I \}$

which results in :

$$\begin{aligned} Rl_g &= \text{agent_on}(P, A), \text{kind}(A, K1), \\ &\text{not_item_on}(P, I); \\ Al_g &= \lgg((\text{kind}(I, \text{Axe}), \text{item_on}(P, I)), \\ &(\text{kind}(I, \text{Sword}), \text{item_on}(P, I))) = \text{kind}(I, K3), \\ &\text{item_on}(P, I). \end{aligned}$$

The reduced $S_g = \text{agent_on}(P, A), \text{kind}(A, K1), \text{not_item_on}(P, I)$, which yields a meaningful generalization. Notice that the rule g matches the example and the rule it was generalized from. \diamond

4.3 Knowledge base management

In the previous section, we discussed how the agent generalized rules from experience. Here, we shall present the management principles of the rule base. The aim of knowledge base management is to memorize only the most interesting rules and to forget the other ones. Forgetting uninteresting rules is necessary for efficiency purposes. Therefore, the knowledge base manager keeps track of a number of counters for each rule:

- tested (T): contains the number of similar actions executed since the rule was

generated;

- applicable (A): contains the number of times the situation part of the rule matched the given situation and the action, executed in the given situation, was similar to the action of the rule;
- correctness (C): contains the number of times the rule's prediction was correct when it was applicable.

One can easily see that the T counter encodes a kind of age of the rule (the number of times it could have been used), and that the probability that the result will be correctly predicted by the rule (when the current situation matches the condition part of the rule and the corresponding action is executed) is C/A . Using these counters, it is easy to define criteria for accepting a rule as promising (and deleting its children from the binary tree) and for deleting a rule. Clearly, rules that have been around for a long time (i.e. with a large T counter) and that are seldom applicable (i.e. with a small A counter) are not interesting as they have a low probability of being used. Also, rules that have a low probability of being correct (i.e. with very low C/A) can best be forgotten. Furthermore, when a rule has been proven to be correct a number of times, its children are forgotten. These principles have been implemented in the system (cf. [Bleken, 1992; Swennen, 1991; Chaouat, 1991]) and have proven to result in small knowledge bases containing useful rules.

5 Learning the weights of an evaluation function

The only direct means to evaluate the performance of the agent is its current score, which is the value of the static evaluation function $Stat(S) = w_1 \times Energy(S) + w_2 \times Gold(S)$ (the weights are known to the agent and w_2 is much smaller than w_1) in the current situation S . Notice that the static evaluation function is not in an operational form, in the sense that

it does not say anything about the relevance of possessing items, of killing opponents, of quaffing potions, etc. When planning for the next action to execute (cf. next section), the agent performs a variant of mini max search using the rules in the knowledge base and an evaluation function. If *Stat* is the evaluation function being used, the agent will have to consider a very deep search tree, which is undesirable: it requires a lot of computation and in the particular context of the game, the leaves become much more uncertain (moving outside the visible part of the board always results in uncertainty as the agent does not know the characteristics of positions outside its view). Therefore *Stat* is not suited as an evaluation function for a planner. Instead, we would like to have an operational evaluation function *Op* that evaluates the current situation in terms of directly observable features F_i , such as the number and kinds of opponents, of (worn and unworn) armors, of (wielded and unwielded) weapons, of wands, of food, etc. A suitable format for such an operational evaluation function would be $Op(S) = \sum_i w_i \times F_i(S)$. In the implemented adventure game, we employ such a function where the numerical features F_i are similar to the ones discussed above. If the operational function is to be relevant to the ultimate goal of the system, as incorporated in the static evaluation function, the two functions should be related. Ideally, we should have that when evaluating $Op(S_1)$ in a situation S_1 it tells us something about $Stat(S_n)$ where S_n is the n -th situation after S_1 ². Ideally, we should have $Stat(S_n) = Op(S_1)$.³

²The algorithm stores the n previous situations and uses the oldest one as S_1 and the most recent one as S_n , n being a parameter of the system.

³At this point, the reader might believe that n should be equal to 2. Whereas an approach where $n=2$ could be followed, many important side effects of actions are not immediate visible. E.g. when wearing an armor, the immediate effect is that the armor is being worn, but the side effect that the agent is now better protected is only noticed later, when the agent is attacked by its opponents. Therefore it is more de-

In general this equation will not be satisfied. When it is not satisfied it is desirable to modify the operational evaluation function, such that it better approximates the static one. Therefore we define the updated evaluation function *uOp*

$$uOp(S_1) = \sum_i w_i \times F_i(S_1) + \sum_i d_i \times (F_i(S_n) - F_i(S_1)) = Stat(S_n)$$

The updated function assumes that the error of the old operational function is due to the parameters that changed in between situation S_1 and S_n . The updated function can be used as the basis of an evaluation function learner. Indeed, we can now compute the updated weights $w'_i = w_i + d_i$ as follows:

$$\begin{aligned} Stat(S_n) &= uOp(S_1) \\ &= \sum_i w_i \times F_i(S_1) + \sum_i d_i \times (F_i(S_n) - F_i(S_1)) \\ &= Op(S_1) + \sum_i d_i \times (F_i(S_n) - F_i(S_1)) \end{aligned}$$

As the equation is underdetermined for the d_i , we have to approximate the d_i making certain assumptions:

- the d_i for which $F_i(S_n) - F_i(S_1) = 0$ are assumed to be 0, reflecting the principle that the weights of features should not be changed without evidence; and
- all non-null features are assumed to have equal impact on the error; therefore $d_i \times (F_i(S_n) - F_i(S_1)) = Ct$ (2) is assumed to be constant for a constant Ct for these features.

Under these assumptions we have:

$$Ct = \frac{\sum_j d_j \times (F_j(S_n) - F_j(S_1))}{m}$$

where j ranges over the m features for which $F_j(S_n) - F_j(S_1) \neq 0$. Together with equations

it is desirable to take slightly larger n , i.e. $n=10$. Too large values for n should be avoided as the effects of the actions disappear when waiting too long.

(1-2), this yields :

$$C_t = \frac{Stat(S_n) - Op(S_1)}{m}$$

$$d_j = \frac{Stat(S_n) - Op(S_1)}{m \times (F_j(S_n) - F_j(S_1))}$$

Our (preliminary) experiments have shown that applying this method as it stands, has two minor problems:

- the weights oscilate frequently and with large values; this can be avoided by replacing m by $m \times f$ where f is a user defined parameter;
- the weights grow steadily and can become very large; therefore it is more appropriate to normalize them after each update.

Using these two changes, the evaluation function learner learns rather quickly. Given 9 features, random play, initial weights being 0 (except for energy and gold), the learning module learns weights with a correct sign (i.e. the direction of the influence of the feature is correct) in less than 300 cycles.

Again, we wish to stress here that the only knowledge being used is similar to that possessed by novice human players: the features that are potentially influence the goal of the game. Furthermore, these features are nearly straightforwardly derived from the knowledge representation. One possible refinement, which we are planning to investigate, is to have a two layered evaluation function, each layer having a format similar to that above; the outer layer would contain the aggregated features such as the number of armors, opponents and weapons, whereas the inner layer would divide each aggregated feature in its components. For armors, this would include the number of helmets, of pairs of gloves, of boots, harness, coats.

6 Planning to survive

Given the evaluation function uOp of the previous section, the knowledge base containing

rules and some objectives, we can use a variant of minimax search to select the most interesting action to execute in a given situation. In adventure games, there are two distinct primary objectives: survival and learning. Indeed the most important objective is definitely to survive as long as possible. However, human players also tend to experiment with several actions, when there are few risks involved, in order to explore the environment and enhance their understanding of the game. Therefore, an intelligent learning agent should also follow this strategy.

In order to plan, the agent starts from a situation description S_0 and a number of actions A_1, \dots, A_n that are executable, i.e. legal, in S_0 . Let us assume that there are rules R_1, \dots, R_k (for each class of rules, only the most general rule considered to be correct, is used for prediction) whose action is similar to one of the actions A_1, \dots, A_j . (For the actions A_{j+1}, \dots, A_n there are no rules in the knowledge base.) Using these rules to predict the outcome of executing action A_i in situation S_0 results in a number of situation descriptions $Sa_{i,1}, \dots, Sa_{i,n_i}$ when executing action A_i ; ... ; in situation descriptions $Sa_{j,1}, \dots, Sa_{j,n_j}$ when executing action A_j . Here, n_i (n_j) is the number of predicted situations when executing action A_i (A_j). Furthermore all situations have an associated probability $p_{s,t}$ defined as the ratio C/A of the rule used in the prediction. Because the sum $\sum_t p_{s,t}$ of these probabilities is not necessarily equal to 1, we normalize them into likelihoods $l_{s,t}$. The estimated value of action A_i is then $E(A_i) = \sum_t l_{i,t} \times E(Sa_{i,t})$. The best action A_i to be executed in situation S_0 for the learning agent is that with a maximum $E(A_i)$; therefore $E(S_0) = \max E(A_i)$. The values of the resulting situation descriptions, the $E(Sa_{i,t})$, are computed similarly. The only differences are that:

- We do not assume that the opponents choose the action with a maximum value for the learning agent, nor the action with

a minimum value, as either of these options would imply that the opponents are aware of the aims of our learning agent and want to help the agent achieve these aims or to make it fail. Rather we assume the opponents act independently of the learning agent and therefore we take the average.

- All of the opponents on the board can execute an action, which affects the current situation. Since we assume all opponents act independently, we define the value of a situation S as the average of the values of situations resulting from the actions of the opponents.⁴

More formally, $E(Sa_{i,t}) = 1/n \sum_l E(O_l)$ ($l = 0 \text{ to } l = n$), where there are n opponents O_l able to execute an action in situation $Sa_{i,t}$; and $E(O_l) = 1/m_l \sum_k E(A_k, O_l)$ ($k = 0 \text{ to } k = m_l$) where there are m_l possible actions A_k in situation $Sa_{i,t}$ that can be executed by agent O_l . $E(A_k, O_l)$ is then computed as for $E(A_i)$ above. Obviously the recursion should terminate at a certain level; this is done by assigning at that, $E(S) = uOp(S)$ for all situations S occurring at that level. Also, situations with a likelihood less than a user defined parameter, are not elaborated further.

The computation of the value of situation S_0 is summarized as follows (see also Figure 3):

- $E(S_0)$ is the maximum value in the set of actions A_j executable by the agent in situation S_0 ;
- the value of an agent's action A_j is the average value of the resulting situations $Sa_{i,t}$, predicted by rules in the knowledge base;
- the value of a situation $Sa_{i,t}$ is the average value of the opponent's O_j in that situation;
- the value of an opponent O_j is the average value of the actions it can execute;

⁴This assumption is not realistic, (cf. Section 7).

- the value of the actions of an opponent, is the average value of the resulting situations.

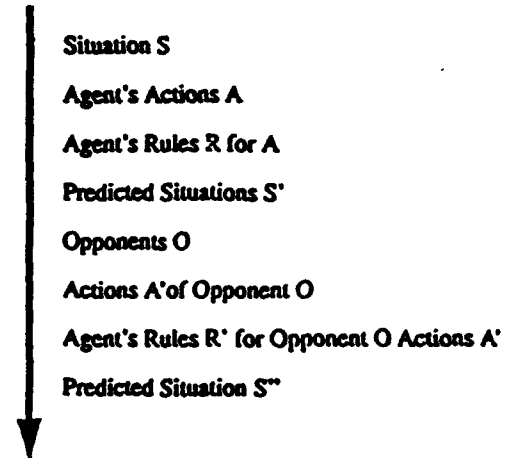


Figure 3: Structure of Search

The planning phase discussed above started from actions A_1, \dots, A_n for which we had rules predicting the actions A_1, \dots, A_j . Planning only estimates $E(A_1), \dots, E(A_j)$; it does not say anything about $E(A_{j+1}), \dots, E(A_n)$. Furthermore, because uOp is used to evaluate the leaves of the search tree, it only takes into account the survival goal (i.e. the aim of the game). To evaluate the interestingness of executing an action A_i with regard to the learning goal, we use the uncertainty of an action (similar to [Scott and Markovitch, 1989] and to the information content frequently used in TDIDT algorithm [Quinlan, 1986]): $U(A_i) = -\sum l_{i,t} \log(l_{i,t})$. The higher the uncertainty of an action, the more interesting it is to execute.

Using these principles, planning for survival and learning is coordinated as follows: if the energy level of the agent is below a user defined critical value, then the agent selects the best action according to the survival objective (i.e. the action A with highest $E(A)$), else

if the best action B according to the learning objective has an uncertainty higher than a user defined critical value, the action B is executed; otherwise an action for which nothing is known is selected (i.e. one of the A_{j+1}, \dots, A_n). Within the above framework, it is straightforward to define alternative strategies (e.g. maximizing survival).

7 Preliminary experiments

The current state of the system is such that all *individual* components are known to work well and also that there is some evidence that the system as a whole functions well:

- the rule generator generates useful rules;
- using an autonomous rule generator and knowledge base management results in a small number of good rules being stored; in one of the experiments (see [Bleken, 1992]), rules for 6 actions were learned from 500 examples; in total about 200 generalizations were computed and the final rule base contained only 10 rules (this means that 690 rules were discarded) of which 2 were examples and the other 8 general and correct rules;
- the evaluation function learner quickly learns evaluation functions that make sense;
- when using good rules, the planner selects appropriate actions, leading to increased energy levels and gold pieces owned and to aged agents (in random play, the agent dies quickly, after about 30 cycles on the average).

Also, the learning system is efficient: to learn and plan, the system (implemented on SUN SPARC using ProLog by BIM) takes a few seconds for each cycle; this time stays approximately constant, also when the knowledge base contains more rules. These preliminary experiments are described in detail in [Swennen, 1991; Chaouat, 1991; Bleken, 1992;

Coget, 1993].

Furthermore, in some preliminary experiments, we were able to show that the system indeed improves its behaviour over time and learns to survive. In these experiments, we started from an empty knowledge base, an evaluation function with weights initialized to 0, and f value of 200, and a minimax tree that predicted one cycle of the game (the effect resulting from the system agent's action and the situations resulting from that by predicting the opponent's actions and effects). The results are shown in Figure 4, and placed in context in Figure 5. In Figure 4, the thin line denotes the score for each of the 11 games. The average score (thin line) of the last games improves slow but steadily. This score is plotted in Figure 5 against the average performance by random play, by novice humans and by expert humans (the designers of the game). Using our multi-strategy learning system, the system performs already better than novice human players. Nevertheless, even after further training, the average score becomes never better than 800-850. Experts perform better. We believe that this is mainly because of some current shortcomings of the system:

- The inability to plan long and complicated sequences of actions; in the hack type of systems it is often desirable to pursue a kind of abstract plan (e.g. move towards food, escape from your opponents, search a particular item, etc.), which may require many actions before these plans are satisfied. Currently the system is shortsighted as it can only look ahead a few steps.
- The inability to compute the combined effects of the opponents. At the moment, the planner computes the average of the situations resulting from the individual actions of opponents, thereby ignoring their combined effects. Two possibilities to avoid the problem include 1) learning more complicated rules that take

into account all agents on the board, and 2) rather than computing the effects of the opponents in parallel, propagate the effects of the opponents, by using sequential prediction.

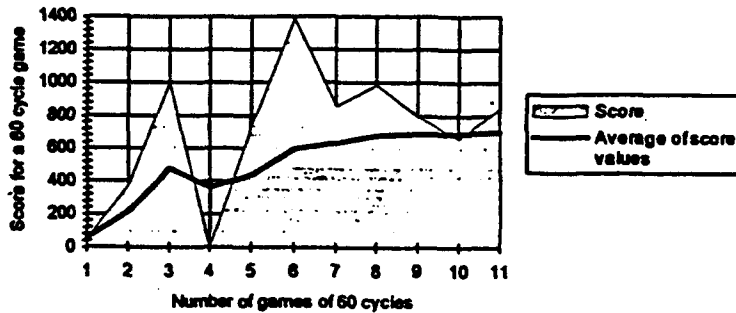


Figure 4: results of learning

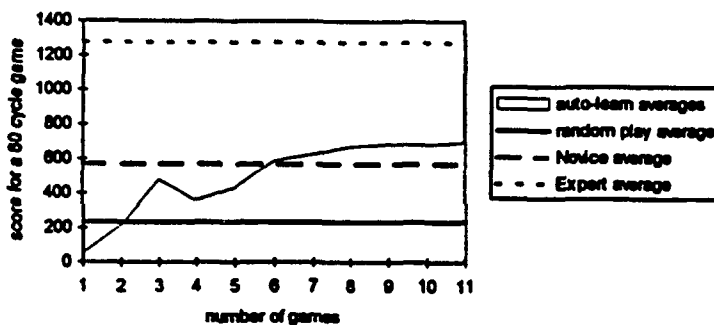


Figure 5: comparison with other results

8 Conclusions and Related Work

We have argued that learning to survive and act in the context of an adventure game is a challenging and realistic task for multistrategy learning, having many interesting features in common with the real-world. We have also outlined an architecture and the components of an autonomous learning agent in such environments. Preliminary experiments indicate that the agent performs quite well. We believe the main reason for this is the integration of a number of learning strategies into a working whole. In particular, our agent uses empirical learning from examples (by applying in-

ductive logic programming techniques) to generate rules, a knowledge base manager to learn from experience, a control module that learns an evaluation function.

To the best of our knowledge, the application of multi strategy learning to learn how to survive is new. Nevertheless, the learning strategies are related to other work. More specifically, the rule generator relies on inductive logic programming [Muggleton, 1992; De Raedt, 1992; Plotkin, 1970] techniques that work from specific to general; the evaluation function learner addresses a problem related to that of credit assignment in the bucket brigade algorithm of [Holland, 1986]; it is also related to the early work of Arthur Samuel [Samuel, 1967]; the knowledge base manager is related to recent approaches in explanation based learning that remember only the most promising rules; and the integration of planning with learning is related to reinforcement learning.

Acknowledgements

The authors are grateful to Alain Callebaut for his help with Prolog by Bim, to Erik Steegmans for help in the initial stage of this project and to Hilde Ade, Gunther Sablon and the referees for suggestions and comments. Luc De Raedt and Maurice Bruynooghe are supported by the Belgian National Fund for Scientific Research and in part by the ESPRIT project no. 6020 on "Inductive Logic Programming".

References

- [Bleken, 1992] E. Bleken. Machine learning in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1992.
- [Chaouat, 1991] G. Chaouat. Towards an autonomous learning agent in an adventure game. Master's thesis, Department of Com-

- puter Science, Katholieke Universiteit Leuven, 1991.
- [Coget, 1993] V. Coget. Learning evaluation functions in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1993. forthcoming.
- [De Raedt, 1992] L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.
- [Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189 – 208, 1971.
- [Holland, 1986] J.H. Holland. Escaping brittleness. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.
- [Muggleton and Buntine, 1988] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.
- [Muggleton, 1992] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.
- [Plotkin, 1970] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5. Edinburgh University Press, 1970.
- [Quinlan, 1986] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [Raymond and Threepoint,] S. Raymond and M. Threepoint. *Nethack guidebook*.
- [Reiter, 1978] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, 1978.
- [Russell, 1989] S.J. Russell. *The use of knowledge in analogy and induction*. Pitman, 1989.
- [Samuel, 1967] A. Samuel. Some studies in machine learning using the game of checkers II - recent progress. *IBM Journal of Research and Development*, 11:601–617, 1967.
- [Scott and Markovitch, 1989] P.D. Scott and S. Markovitch. Learning novel domains through curiosity and conjecture. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 669–674. Morgan Kaufmann, 1989.
- [Stephenson,] M. Stephenson. *Nethack: the unix implementation of the game*.
- [Swennen, 1991] B. Swennen. Learning by discovery in an adventure game. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1991.

Appendix 1: the game

List of opponents: Humans H, Dragons D and Bats B.

List of objects:

- Gold \$: needed to become rich
- Food %: needed to increase the energy level, consuming cursed food decreases the energy level however,
- Potions !: quaffing potions adds or removes curses from an object of the same type,
- Armors [: protect against attacks by opponents,
- Handweapons]: wielded handweapons are needed to hit the opponents,
- Wands /: change the current situation of the game; wands of death kill the agent at which it is zapped, wands of teleport change the current situation randomly (except for obstacles), wands of polymorph change the type of object or opponent at which it is zapped.

- Spells [?]: change the current situation of the game; killing spells have the same effect as hitting all opponents simultaneously; teleport spells change the current situation randomly; and cursing spells add or remove curses from objects.

List of Actions:

- move(position): moving to a neighbouring position which is not an obstacle or opponent.
- pick_up(item): pick up an item on the current position.
- drop(item): drop an item currently possessed.
- eat(food): eat food currently possessed.
- hit(position): hit an opponent on a neighbouring position. The hitting agent loses energy and the opponent may but need not lose energy.
- wield(handweapon): wield a handweapon currently possessed.
- release(handweapon): release a handweapon currently possessed.
- wear(armor): wear an armor currently possessed.
- take_off(armor): take off an armor currently worn.
- cast(spell): cast a spell currently possessed. Spells can be cast only once.
- quaff(potion): quaffing a potion currently possessed. Potions can be quaffed only once.
- zap(wand): zap a wand (currently possessed) to a neighbouring position. Wands can be zapped only once.
- Agent_on (pos, agent);
- Item_on (pos, item);
- Obstacle_on (pos);
- Possessing (item);
- Wielding (item);
- Wearing (item);
- Cursed_by (item, curse) (only for items possessed by learning agent);
- Un_Cursed (item) (only for items possessed by learning agent);
- Alive (learning agent is alive);
- Dead;
- My_Kind (kind) (learning agent is of type kind);
- My_Energy (energy) (energy level of learning agent);
- My_Gold (gold);
- My_Curse (curse) (learning agent is cursed);
- Un_Cursed;
- Kind_of_Agent (agent, kind);
- Kind (item, kind); possible kinds are : gold, edible, armor, spell, wand, weapon.
- Kind_of_Armor (item, kind); possible kinds are: boots, harness, helmet, coat, gloves.
- Kind_of_Weapon (item, kind); possible kinds are: hand_weapon, wand.
- Kind_of_Edible (item, kind); possible kinds are: food, potion.
- Kind_of_Hand_weapon (item, kind); possible kinds are: sword, dagger, axe.
- Kind_of_Wand (item, kind) possible kinds are: wand_of_polymorph, wand_of_teleport, wand_of_death.
- Name_of_spell (item, name);
- Curse_of_potion (item, curse);
- No_Agent_on (pos);
- Not_Item_on (pos, item);

Appendix 2: representations of situations

The following predicates are used to describe situations, rules and examples.

- No_Obstacle_on (pos);
- Not_Possessing (item);
- Not_Wielding (item);
- Not_wearing_such_an_armor (item);
- Not_Cursed_by (item, curse);
- Not_My_Curse (curse);

The literals starting with Not and No are the negations for the corresponding positive literals. To reason about the energy and the gold of the system agent, there are some elementary literals for comparison such as less_than, equal_to and less_than_or_equal_to.

MUSKRAT: a Multistrategy Knowledge Refinement and Acquisition Toolbox

Nicolas Graner and D. Sleeman
 Department of Computing Science
 University of Aberdeen
 Aberdeen AB9 2UE
 Scotland, UK

Abstract

This paper presents MUSKRAT, a Multistrategy Knowledge Refinement and Acquisition Toolbox. MUSKRAT is an open architecture which supports the integration of problem solvers and various types of knowledge acquisition tools, including knowledge elicitation, machine learning, and knowledge base refinement tools. All the knowledge acquired is expressed in a Common Knowledge Representation Language (CKRL), and can be shared by several problem solvers; each tool translates its internal knowledge representation formalism to or from CKRL. An advice-giving system compares the requirements of the selected problem solver with available sources of information (knowledge, data, human expert...) and recommends one or more knowledge acquisition tools, based on a knowledge-level description of each tool. We describe the MUSKRAT architecture, and illustrate it with a detailed description of a prototype currently being implemented, which includes three problem solvers and four knowledge acquisition tools.

Key words: Knowledge acquisition, Integrated system, Machine learning, Knowledge elicitation, Knowledge base refinement

1 Introduction

Research into knowledge-based systems originally focused on building inference engines. It then became progressively clear that the most significant bottleneck was not in the inference engine but in the acquisition of knowledge. After considerable experience of carrying out this process, one valuable insight was that knowledge based systems which attempt to address the same sort of task have much in common, and that once the type of problem solver was determined it was much easier to decide what domain knowledge was required. Several researchers have attempted to build taxonomies of problem solving (Clancey, 1985; Hayes-Roth, 1983), and others suggested that specific tools should be built to acquire knowledge for each problem solving method (McDermott, 1988).

Many tools and techniques have been developed for the systematic acquisition of domain

knowledge, including knowledge elicitation (KE) methods to acquire knowledge from a human expert, machine learning (ML) algorithms that infer knowledge from data, and knowledge base refinement (KBR) tools that refine knowledge already in a usable form. As these tools become more sophisticated and are enhanced to deal with real-world applications, their differences tend to become less apparent. KE tools, which were originally simple implementations of manual methods, now perform more tasks automatically, while ML and KBR tools now recognise that many interesting tasks cannot be completely automated and so interact more with their users. This, in addition to the large number of available Knowledge Acquisition (KA) techniques, makes it very difficult for many users to choose an appropriate tool for their particular application, especially when more than one is needed to solve their problem.

Our aim is to relate the several types of problem solvers, and hence the kinds of knowledge that they require, with these tools. We would then be able to give advice on what tools should be used to acquire, transform or refine the knowledge so it can be used in a particular problem solver.

This paper presents MUSKRAT, a MultiStrategy Knowledge Refinement and Acquisition Toolbox. MUSKRAT is an open architecture which supports the integration of problem solvers and KA tools, and assists the user with the selection of the most suitable KA tool. Section 2 introduces some motivations for MUSKRAT in relation with other work; section 3 describes the MUSKRAT architecture; section 4 discusses the various problem solvers and KA tools included in the prototype that we are currently developing.

2 Background

This work originated in the Machine Learning Toolbox (MLT) project. The aim of this project was to bring machine learning into use on real industrial problems, by (among other tasks) building a collection of ML tools. This toolbox also includes a number of support tools, including an advice-giving system, the Consultant, which we developed at the University of Aberdeen (Craw, 1992). The Consultant questions its user about the task he wants to solve, the data and background knowledge he can provide, etc., and recommends one or more suitable learning tools. Although it was found to perform satisfactorily, the Consultant suffers from a major limitation: it has no understanding of the problem that the user wants to solve in his application domain. The user must first decide what knowledge is required to solve his problem, i.e. define a learning task, and only then can the Consultant help him with the choice of a suitable tool. In other words, the Consultant is told what the user wants to know, not what he wants to do.

Having a model of the target problem solver would be useful, not only to help the user specify his learning task, but also to guide the KA process itself. This is generally acknowledged in the KE community (McDermott, 1988).

"Currently the main theories of knowledge acquisition are all model based to a certain extent. The model based approach to knowledge acquisition covers the idea that abstract models of the tasks that expert systems have to perform can highly facilitate knowledge acquisition." (van Heijst, 1992)

However, this is not always accepted by ML researchers, and ML systems that use an explicit model of problem solving are rare (Ganascia, 1993). The reason is that an abstract, knowledge-level model of a problem solver is usually not sufficient to guide ML effectively. The knowledge acquired by "manual" KE can easily be adapted (if necessary) so as to fit the requirements of a particular problem solver (e.g. in terms of knowledge representation), but that obtained from an automatic ML tool can only be used directly if the detailed needs of the problem solver are known in advance. Instead, the requirements are usually assumed and encoded implicitly in the ML tool. This is even clearer with knowledge base refinement tools, which must generally run a problem solver on the knowledge they refine in order to evaluate their modifications.

For these reasons, we decided that MUSKRAT, a knowledge acquisition toolbox which includes KE, ML and KBR tools, should also include problem solving tools, which will serve as the targets of the KA process. This is in contrast, for instance, with KEW, the Knowledge Engineering Workbench produced by the ACKnowledge project (Reichgelt, 1992). Since KEW focuses on KE techniques, it does not include a problem solver, but instead uses Generalised Directive Models to guide tool selection (van Heijst, 1992).

The integration of learning and problem solving is also a major issue in the field of integrated systems (SIGART, 1991; VanLehn, 1991). Some such systems integrate one or several KA tools with a problem solver (as in PRODIGY (Carbonell, 1991)). Others integrate KA and problem solving in a single component, using a uniform technique (THEO

(Mitchell, 1992), SOAR (Laird, 1991)). In both cases, the knowledge base is tied to a particular problem solver. In contrast, MUSKRAT integrates existing, stand-alone KA tools with existing, stand-alone problem solvers, so that the knowledge can be tested independently and shared among several problem solvers. Knowledge sharing and reuse is further supported by the fact that all the knowledge acquired by the system is expressed in a single representation language, called CKRL.

Finally, the selection of an appropriate KA tool is also an issue in many multistrategy learning systems (Michalski, 1991). Some multistrategy systems include several ML techniques (for instance both symbolic and sub-symbolic algorithms) which are applied successively to generate a single knowledge base. In MUSKRAT, the knowledge to be acquired is structured into several knowledge bases, each of which is obtained with an appropriate technique. Other multistrategy systems include several similar techniques and use highly discriminating selection criteria to select the most suitable one, but we are not aware of any system that covers as broad a range of techniques as MUSKRAT, including KE, ML and KBR tools.

3 The MUSKRAT Architecture

The acquisition of control knowledge (i.e. a problem solving method) and domain knowledge can be performed in any relative order. In the MUSKRAT framework, we assume that the selection of one or several KA techniques proceeds along the following lines:

1. Identify an *application task*, i.e. a problem to be solved in a particular domain.

2. Select a suitable problem solver to solve this task. If no single problem solver can be identified, it may be necessary to split the application task into sub-problems that can each be solved by a problem solver.
3. For each selected problem solver, determine what knowledge bases are required. This amounts to a knowledge-level analysis of each problem solver, which needs only be done once since it does not depend on a particular application task.
4. For each required knowledge base, compare the problem solver's requirements with whatever knowledge sources are available (human expert, examples, existing knowledge, etc.). This defines one or more *KA tasks*.
5. Select a KA tool capable of solving each KA task, i.e. bridging the gap between required and available knowledge. This supposes a preliminary knowledge-level analysis of available tools.
6. Apply the selected KA tool.

These steps can be repeated in a cycle, especially if information acquired in step 6 is needed to refine the decisions made in step 2.

The MUSKRAT system is designed to support steps 3 to 6. It assumes that a problem solver has been selected for a particular task or sub-task, and directs the acquisition of knowledge for this particular problem solver. The system consists of any number of problem solvers, any number of KA tools, and a guidance module, the KA selector.

The architecture is centred around a set of Knowledge Bases (KBs), which is the inter-

face between KA tools and problem solvers. We define a KB as any body of knowledge required by a problem solver. This includes not only "conventional" knowledge bases (e.g. rule sets), but also representation languages, control heuristics, etc.

In MUSKRAT, all KBs are expressed in the same representation language, CKRL. CKRL (Common Knowledge Representation Language) is an information interchange language developed as part of the MLT project (Morik, 1991). It is not directly executable, but consists of declarations that can be translated into a tool's internal representation. To ensure that this translation is possible into a broad range of representation languages, and unambiguous, CKRL entities are defined at the *epistemic* level (Brachman, 1979): concepts, instances, relations, properties, sorts, rules, etc. Although CKRL was originally designed as a communication medium for ML tools, it is general enough to be useful in many situations where knowledge is to be transmitted or processed in a number of ways.

Our choice of a uniform knowledge representation was motivated by considerations of knowledge sharing and reuse: a KB can be used by several problem solvers, even if this was not anticipated when the KB was created. It also allows the integration of new problem solvers and KA tools into MUSKRAT at the cost of implementing a single interface to or from CKRL. An additional advantage of choosing CKRL is that some of the KA tools in our prototype are also part of the MLT, and therefore already express their output in this language (see section 4.2).

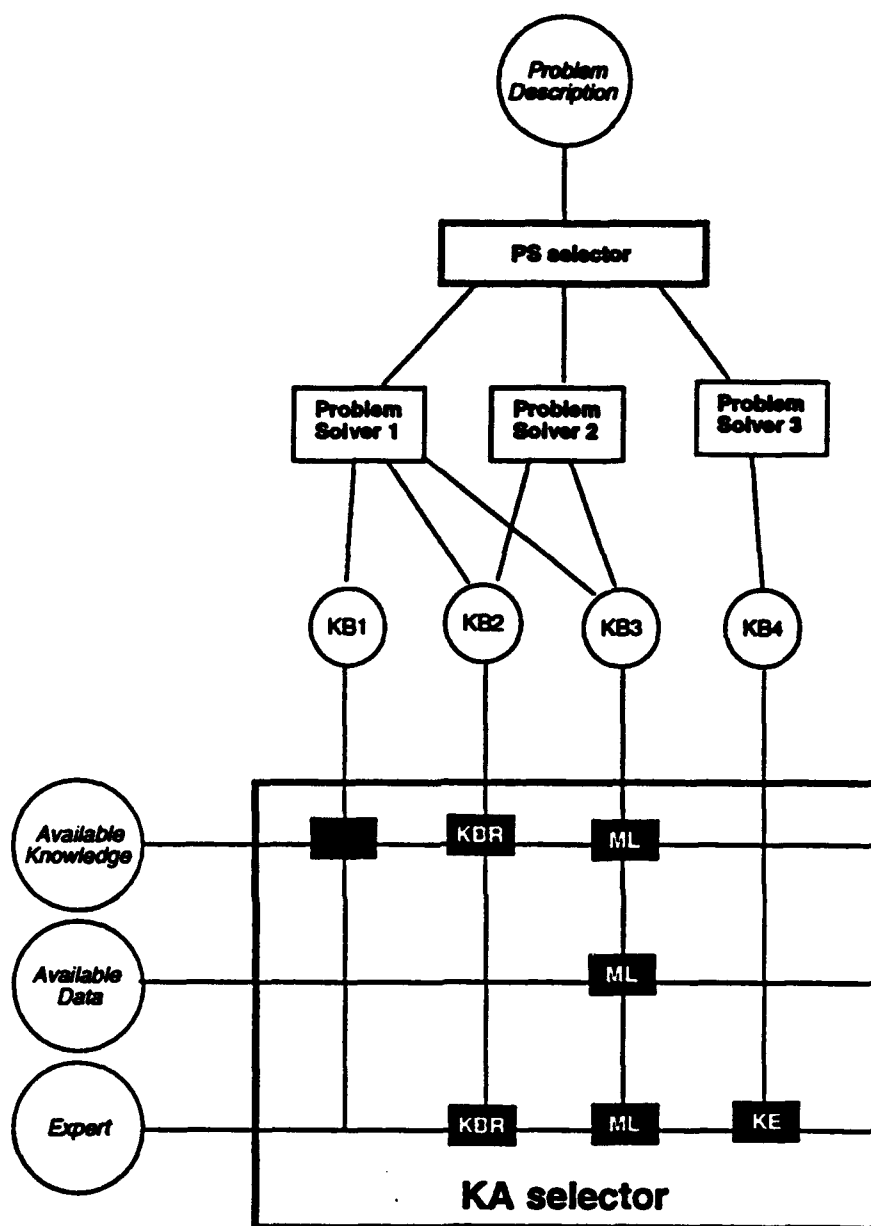


Figure 1: Overview of the MUSKRAT architecture.

In figure 1, circles represent bodies of knowledge and boxes represent MUSKRAT's components. There are three types of boxes: thin boxes represent problem solvers, black boxes represent KA tools, and thick boxes represent advice-giving systems. The types of knowledge provided to MUSKRAT are an initial problem description (top) and various sources of problem solving knowledge (bottom left), and the system is used to produce one or more KBs (middle).

This architecture can be used at two different stages of the problem solving cycle: (a) the selection of suitable tools, and (b) the use of the tools to acquire knowledge and solve the problem. We will now describe these two stages in detail, and explain the role of each component.

The tool selection process starts with an initial description of the problem. An advice-giving system, the PS selector, helps the user with the selection of a suitable problem solver. The

PS selector is not currently part of MUSKRAT, but a module similar to KEW's advice and guidance module (van Heijst, 1992) should be usable here.

Once a problem solver has been selected, MUSKRAT knows which KB(s) are required. For instance, in figure 1, if we want to use the first problem solver (Problem Solver 1) we will have to acquire three KBs: KB1, KB2 and KB3. At this stage, each KB is only specified in terms of its functionalities and representation, as required by the problem solver. These requirements are expressed in a formalism which provides descriptors for both knowledge-level and symbol-level features. We are currently defining such a formalism to describe the effects and requirements of the particular tools included in the MUSKRAT prototype (section 4). Since these tools cover a fairly broad range of techniques, we expect that our formalism will be easily extended/refined to be applicable to most existing KA techniques.

The next step is to identify the available knowledge sources (bottom left in the figure). We consider three broad categories of knowledge sources: *available knowledge* refers to knowledge that is already in the form required for a KB, e.g. a set of rules. It may be directly usable or require further transformation or refinement. Note that knowledge is seldom available initially, but when MUSKRAT is used iteratively as part of a problem solving cycle, "available knowledge" refers to that acquired during a previous iteration. *Available data* refers to data that is relevant to the problem and from which useful information could be extracted, although it does not meet the requirements of the KB. Typically, this may consist of past cases, i.e. previously solved problems similar to the one at hand, from

which insight into the new problem can be gained. Alternatively, if the problem is to diagnose faults in a complex system, "available data" may refer to a model of the system, which is useful (or perhaps necessary) to perform diagnosis. Note that the distinction between knowledge and data is not intrinsic but depends on the KB requirements. For instance, a set of past cases is considered as knowledge if it is to be used by a case-based reasoner that can use it directly, but it is only data for a rule-based system which is unable to reason from cases. Finally, an *expert* is a person who can provide various forms of knowledge, possibly with the help of a KE tool and/or a knowledge engineer.

The KA selector is the central component of MUSKRAT. It compares the requirements of the selected problem solver with the characteristics of available knowledge sources and recommends the use of one or more KA tools. For that purpose, it has a knowledge-level description of each available KA tool and performs a means-ends analysis to decide which one is most capable of reducing the differences.

Since there are three types of knowledge sources that the KA selector can decide to use or not use, eight (2^3) combinations could be considered. For each combination, a suitable KA technique must be identified. The four most common combinations are represented in figure 1 by vertical lines ending in each of the four KBs:

1. Available knowledge exactly matches required knowledge, and can thus be used directly, or possibly with some syntactic data manipulation (not represented). This is shown as an empty black box, representing direct transfer of knowledge.

2. Available knowledge can be used by the selected problem solver, but does not produce satisfactory results (this normally occurs when MUSKRAT is used iteratively, and the knowledge in question was obtained and tested during a previous iteration). A KBR tool can then be used to refine this knowledge. A domain expert is usually required, either during the refinement process or only to validate the resulting KB
3. If there is not enough or no available knowledge but other data is present, an ML tool can be used to extract useful knowledge. Depending on the selected tool, existing (incomplete) knowledge may also be used, and an expert may be required to interact with the ML tool. In any case, an expert is almost always required to validate the newly acquired KB.
4. If no other source is available, the KA selector still has the resource to recommend a KE tool, which will attempt to obtain the required knowledge directly from an expert, using a more or less systematic methodology.

Another plausible situation (not illustrated) is that no tool is available to produce the required KB. In this case, the KA selector will merely describe the requirements to an expert and let him provide this knowledge "manually". The help of a knowledge engineer will usually be necessary in this case; it is also desirable in the previous cases.

The tool selection process is summarised in figure 2, where thick arrows represent the flow of information which converges from two different directions into the KA selector.

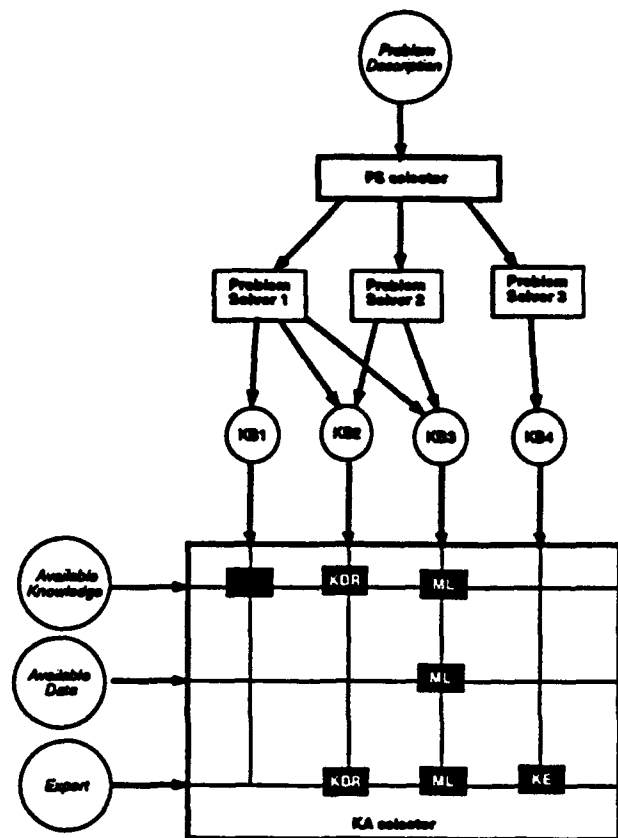


Figure 2: Flow of information for tool selection.

Once appropriate problem solvers and KA tools have been selected, they can be used to actually solve the problem. Since MUSKRAT only performs the integration of independent tools, it provides no support with the use of individual tools. At this stage, its role is limited to the communication of knowledge between different tools. The flow of information, illustrated in figure 3, shows knowledge coming out of knowledge sources, being processed by KA tools, and finally used by a problem solver. It is the user's responsibility to evaluate the solution obtained by the problem solver and, if necessary, to start a new KA cycle.

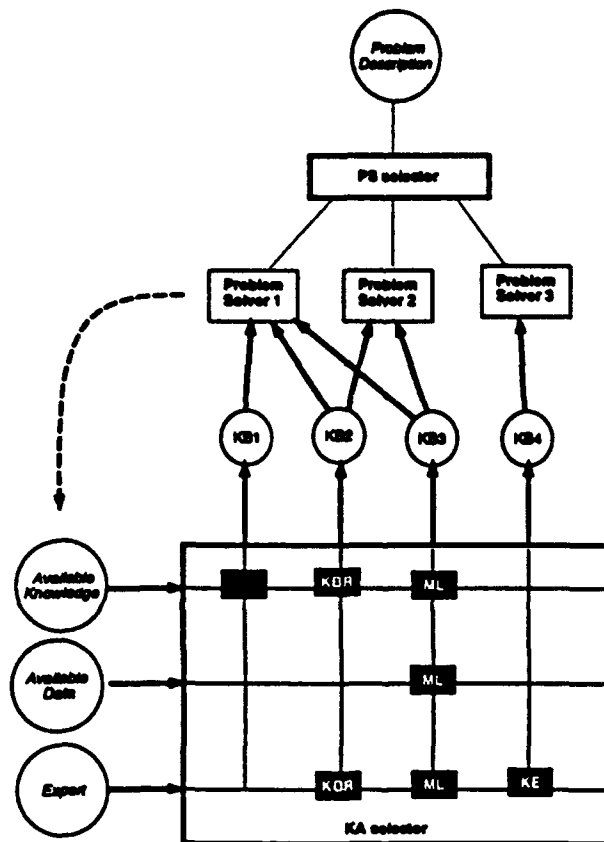


Figure 3: Flow of information for problem solving.

4 The MUSKRAT Prototype

To illustrate our approach, we have chosen a "toy" domain where several problems can be identified which require different problem solvers, but where some of the KBs required can be shared by at least two problem solvers. We are currently developing a prototype toolbox including three different problem solvers and four KA tools, selected to suit these particular problems. All these tools are described in more detail below.

It should be noted that, since the main focus of our work is on knowledge acquisition rather than problem solving, we decided to keep the problem solvers fairly simple, even if this implies that we can only solve simplified versions of our original problems.

4.1 Problems and problem solvers

The domain that we consider is the planning of a meal. It includes three distinct problems: selecting dishes given a set of constraints, analysing and criticising a selected menu, and scheduling the meal preparation given time constraints and limited resources.

Enhanced version of these problem solvers will later be applied to similar, though much larger, problems in the domain of flexible manufacturing, namely the customised design of mechanical devices under specific constraints, analysis of existing designs, and flexible workshop scheduling.

4.1.1 Constraint satisfaction

The problem can be described as follows: given a set of constraints, select a menu (from a pre-defined list of dishes) that satisfies the largest number of constraints. Examples of constraints include: "the meal should include a starter, a main course and optionally a dessert", "select a vegetarian meal", "at most one dish may include sea food", "the total price must not exceed N ", etc.

If all the constraints cannot be met simultaneously, the system must decide which constraint(s) should be relaxed first.

The KBs required to solve this problem are:

- A1. A set of descriptors (attributes) used to represent dishes. We only use boolean- and numeric-valued descriptors. Examples include "has-meat", "warm", "is-starter", "cost", etc.
- A2. A set of dish descriptions. A description consists of a list of ingredients, and the

values of general descriptors such as "is-soup" or "warm".

- A3. A set of rules that infer the values of descriptors from ingredients and/or other descriptors. These rules are used to generate complete internal representations of dishes from incomplete user-provided descriptions. Examples include "beef \in ingredients \Rightarrow has-meat", "has-meat \Rightarrow \neg vegetarian" and "cost $> 5 \Rightarrow$ expensive".
- A4. Predefined constraints that can be used in queries, such as "vegetarian-meal \equiv all(is-vegetarian)" or "cheap-meal \equiv sum(cost) $< \pounds 6$ ". A query can combine any number of predefined constraints and user-defined constraints written using a fixed set of operators.
- A5. Meta-rules that tell the system which constraint to relax when all the constraints cannot be satisfied simultaneously. For instance, a rule might say that cost-related constraints are less important than dietary constraints, or that the number of courses is the most important constraint.

4.1.2 Design analysis

The purpose of this problem solver is to take a menu generated by the first problem solver or any other source, and to issue a list of comments, and suggestions for possible improvements. A typical output from this module could be "This meal supplies 2/3 of the recommended daily allowance of carbohydrates", or "This meal is unbalanced because it contains two sea food dishes; you should replace fish soup with vegetable soup".

The KBs required by this problem solver are:

- B1. A set of dish descriptors (identical to A1).
- B2. A list of all the dishes that may appear in menus (identical to A2).
- B3. A set of description expansion rules (identical to A3).
- B4. A set of rules that derive comments and recommendations from descriptors, for instance "count(seafood) $> 1 \Rightarrow$ comment(too-much-seafood)". Each comment is associated with a canned piece of English text.

4.1.3 Task scheduling

Once a menu has been selected, this problem solver can be used to generate a plan to prepare it. Each dish has a recipe, which is a fixed, partially ordered list of actions. The problem is to set the starting time of the tasks involved in the recipes of all the dishes in the menu, so as to meet a set of time and resource constraints. A time constraint may be that two dishes must be ready and warm at the same time; a resource constraint may be that only one oven is available, therefore at most one dish can be baked at a time.

The KBs required by this problem solver are:

- C1. A list of dishes with associated recipes. This is a superset of A2, since a recipe includes a list of ingredients and an ordered list of actions.
- C2. A list of available resources, and the amount of each resource that is available. These resources are supposed to be permanent and usable for any number of

tasks (e.g. an oven); resources that are consumed by a particular task are listed as "ingredients". Ingredients are always assumed to be available in sufficient amounts, since they have to be purchased for each dish.

4.2 Knowledge acquisition tools

Unlike the above problem solvers, which are being implemented as part of this work, the KA tools described in this section were developed independently by other researchers. Our goal is to show that they can be made to work together to produce complementary KBs, with minimal modifications. One significant enhancement that has to be made, however, is that they must all express their output in the same knowledge representation language, CKRL. This is already the case for those which are part of the MLT, namely APT and KBG.

4.2.1 Repertory grid

The repertory grid is a KE technique derived from cognitive psychology (Kelly, 1955). It provides a systematic way of interactively eliciting *elements* (examples) and *constructs* (descriptors) from an expert. Although it is fundamentally a methodology, it can be supported by software tools such as Tactix (Reichgelt, 1992) that not only acquire this knowledge but also compute similarities and correlations between elements and between constructs.

In our application, this tool is used to acquire simultaneously dish descriptors (A1 or B1) and descriptions (A2 or B2), since these KBs must be acquired directly from an expert. In addition, correlations between descriptors suggest possible rules for A3 (or B3), al-

though those can be more adequately acquired by KBG (see below).

4.2.2 KBG

KBG (Bisson, 1992) is an ML clustering and generalisation tool. It can either take unclassified examples and cluster them according to a particular, flexible metric, or take classified examples and induce discrimination rules. In both cases, it can also use background knowledge in the form of rules to complete example descriptions. An interesting feature of KBG is that its learning examples and output rules are expressed in (restricted) first-order logic, which means in particular that all the examples need not be represented by the same descriptors.

In our example, KBG is used to infer rules for A3 (and B3): given a small number of complete dish descriptions, it finds correlations between descriptors that can later be used to complete new (incomplete) descriptions.

It can also be used to infer control rules for A5. In this case, a learning example is a set of constraints and an indication of which one should be dropped. Since constraints are complex objects that cannot be represented as attribute/value pairs, KBG's first order representation is very suitable for this task.

Finally, its clustering and concept formation ability can be used to select useful predefined constraints (A4). Since such constraints are provided only for user convenience, it is useful to detect patterns that occur frequently in user-defined constraints, and add them to the set of pre-defined constraints. KBG can help with this pattern detection.

4.2.3 APT

APT (Nédellec, 1992) uses a combination of KE and ML techniques to acquire problem solving rules. It starts with a domain theory, in the form of a semantic network, and possibly an initial set of rules. When it cannot solve a problem with its rules, it asks the user for a particular solution, then uses the domain theory to generalise it. The user is constantly requested to validate the rules generated by the system, and he can extend the domain theory if necessary to enable APT to infer correct generalised rules.

When a limited domain model is available, APT can be used as a KE tool to acquire rules and enhance the model. When an important set of rules is available, it can be seen as an interactive KBR tool. These capabilities, together with its rich knowledge representation (semantic network) make it suitable to acquire analysis rules (B4), as well as recipes (C1) that can be regarded as problem decomposition rules.

4.2.4 KRUST

KRUST (Craw, 1990) is an automatic KBR tool. Given a set of Prolog-like rules, and an example incorrectly classified by these rules, it considers many possible remedies (generalising or specialising rule premises, re-ordering rules, adding new rules...), tests them against known cases and implements the most successful ones. It occasionally consults an expert to validate its recommendations.

In our prototype, KRUST uses examples of menus commented on by an expert to refine the "comment" rules (B4).

4.3 Summary

The following table summarises the relationships between MUSKRAT's components. For each knowledge base, it shows which KA tools can generate it and which problem solvers can use it.

KB	created by	used by
A1, B1	Grid	const. sat., analysis
A2, B2	Grid	const. sat., analysis
A3, B3	KBG, Grid	const. sat., analysis
A4	KBG	const. sat.
A5	KBG	const. sat.
B4	APT, KRUST	analysis
C1	APT	scheduling
C2	(no tool)	scheduling

5 Conclusion

We have presented an architecture that allows the integration of independent problem solving and knowledge acquisition tools into a uniform framework. Integration is achieved by means of two common languages: a knowledge-level description of the tools (used by the KA selector to provide advice and guidance), and a uniform representation of data (which encourages knowledge sharing and reuse). A prototype is currently being implemented to validate this architecture. In particular, we will need to evaluate the extensibility of the system. When a new tool is added, a knowledge-level description of its functionality must be provided. It is at present difficult to estimate the effort required to derive such a model. A challenging goal for future research would be to generate, or, more realistically, to refine, knowledge-level models of problem solvers by letting the system perform its own experiments. This

opens interesting perspectives in the field of autonomous multistrategy learning systems.

Acknowledgements

This work is partially supported by the Commission of the European Communities as part of the Machine Learning Toolbox (MLT) project (ESPRIT project P2154). The partners involved in this project are Alcatel Alsthom Recherche (France), British Aerospace (UK), Foundation of Research and Technology -- Hellas (Greece), Gesellschaft für Mathematik und Datenverarbeitung (Germany), INRIA (France), ISoft (France), Siemens AG (Germany), The Turing Institute (UK), Universidade de Coimbra (Portugal), Université de la Réunion (France) and Université de Paris-Sud (France).

We are indebted to Maarten van Someren for several interesting discussions and comments. Our current work on a formalism for describing KB requirements is largely inspired by his own research.

We would also like to thank Nigel Shadbolt and three anonymous reviewers for their comments on an earlier draft of this paper.

References

- Bisson, G., "Learning in FOL with a Similarity Measure, in *Proc. 11th National Conference on Artificial Intelligence*, July 1992.
- Brachman, R. J., "On the epistemological status of semantic networks", in *Associative Networks: Representation and Use of Knowledge by Computers*, Findler, N.V., (Ed.), Academic Press, Orlando, 1979.
- Carbonell, J.G., Knoblock, C. A., and Minton, S., "Prodigy: An integrated architecture for planning and learning", in (VanLehn, 1991).
- Chandrasekaran, B., "Towards a taxonomy of problem solving types", *AI Magazine*, Vol. 4, Winter/Spring 1983.
- Clancey, W. J., "Heuristic classification", *Artificial Intelligence*, Vol 27, No 3, 1985.
- Craw, S., and Sleeman, D., "Automating the refinement of knowledge-based systems", in *Proceedings of ECAI 90*, Aiello, L. C. (Ed), Pitman, 1990.
- Craw, S., Sleeman, D., Graner, N., Rissakis, M., and Sharma, S., "CONSULTANT: Providing advice for the Machine Learning Toolbox", in *Proceedings of the 1992 BCS Expert Systems Conference*, Bramer, M. (Ed), Cambridge University Press, 1992.
- Ganascia, J.-G., Thomas, J., and Laublet, P., "Integrating models of knowledge and machine learning", in *Machine Learning: ECML-93*, Brazdil, P.B. (Ed), Springer-Verlag, 1993.
- Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., (Eds), *Building expert systems*, Teknowledge series in knowledge engineering, volume 1, Addison-Wesley, Reading, Mass., 1983.
- Kelly, G. A., *The Psychology of Personal Constructs*, Norton, New York, 1955.
- Laird, J., Hucka, M., Huffman, S., and Rosenbloom, P., "An analysis of Soar as an integrated architecture", in (SIGART, 1991).
- McDermott, J., "Preliminary steps towards a taxonomy of problem solving methods", in *Automating Knowledge Acquisition for Expert*

Systems, Marcus, S. (Ed), Kluwer Academic Publishers, 1988.

Michalski, R. S. and Tecuci, G. (Eds), *Proceedings of the First International Workshop on Multistrategy Learning (MSL-91)*, George Mason University, Fairfax, VA, November 1991.

Mitchell, T.M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., and Schlimmer, J. C., "Theo: A framework for self-improving systems", in (VanLehn, 1991).

Morik, K., Causse, K., and Boswell, R., "A common knowledge representation integrating learning tools", in (Michalski, 1991).

Nédellec, C. and Causse, K., "Knowledge refinement using Knowledge Acquisition and Machine Learning Methods", in *Proceedings of EKAW*, Springer Verlag, May 1992.

Ralambondrainy, H., Demonchaux, E., and Jomier, G., "Data analysis, data bases and expert systems: the common interface", in *Astronomy from Large Databases: Scientific Objectives and Methodological Approaches*, Murtagh and Heck (Eds), ESO Conference and Workshop Proceedings No. 28, Oct. 1987.

Reichgelt, H. and Shadbolt, N., "ProtoKEW: A knowledge-based system for knowledge acquisition", in *Artificial Intelligence*, Sleeman, D. and Bernsen, N. O. (Eds), Research Directions in Cognitive Science: European Perspectives, volume 6, Lawrence Erlbaum, Hove (UK), 1992.

SIGART Bulletin, Vol 2, No 4, Special section on integrated cognitive architectures, Aug. 1991.

Van Heijst, G., Terpstra, P., Wielinga, B., and Shadbolt, N., "Using generalised directive models in knowledge acquisition", in *Proceedings of EKAW*, Springer Verlag, May 1992.

VanLehn, K. (Ed), *Architectures for Intelligence (Proceedings of the 22nd Carnegie Mellon Symposium on Cognition, 1988)*. Lawrence Erlbaum, Hillsdale, NJ, 1991.

Wielinga, B. J., Schreiber, A. T., and Breuker, J. A., "KADS: a modelling approach to knowledge engineering", *Knowledge Acquisition*, Vol 4, No 1, March 1992.

III. Cooperative Integration

Plausible Explanations and Instance-Based Learning in Mixed Symbolic/Numeric Domains

Gerhard Widmer

Department of Medical Cybernetics and Artificial Intelligence, University of Vienna, and
Austrian Research Institute for Artificial Intelligence,
Schottengasse 3, A-1010 Vienna, Austria
e-mail: gerhard@ai.univie.ac.at

Abstract

The paper is concerned with supervised learning of numeric target concepts. The task is to learn to predict or determine the exact values of some numeric target variables. Training examples may be described by both symbolic and numeric predicates. General domain knowledge may be available in qualitative form. The paper presents a general learning model for such domains. The model integrates a symbolic learning component, which is based on a multi-instance plausible explanation algorithm, and an instance-based learning component, which stores instances with precise values and predicts new values by interpolation. The symbolic component can use available qualitative background knowledge; it learns sub-concepts that partition the space for the underlying instance-based method. A realization of the model in a system named IBL-Smart is then described. The system has been applied to a complex task from the domain of tonal music, and some experimental results are reported that demonstrate the effectiveness of the method.

Key words: Knowledge-based learning, instance-based learning, integrated learning, qualitative models.

1 Introduction

It is being recognized by more and more researchers that qualitative background knowledge is naturally available in many domains, and that learning algorithms are needed that can effectively use such knowledge, even if it is incomplete and inconsistent, and generally abstract and imprecise. Some approaches to this problem have been proposed in the recent past, most of them centering around the notion of incomplete or plausible explanations (see, e.g., Tecuci, 1991; Tecuci & Michalski, 1991; Widmer, 1991). All these methods and systems assume that the target concepts are discrete classes of objects, to be described by classification rules which assign a new object to its appropriate class.

However, there are also many learning problems with numeric target concepts, i.e., where the task is to predict more or less precisely the values of some numeric variables. The training instances may be described by both symbolic and numeric predicates. In such domains, too, general domain knowledge may be available

that relates certain parameters, but maybe only in a qualitative, imprecise way. As an example, consider typical prediction tasks such as stock market prediction or the prediction of energy consumption or demand in some power plant. One can easily conceive of partial qualitative models of these domains that would capture some of the relevant domain knowledge. Intelligent learners should be able to utilize such abstract knowledge.

With few exceptions (e.g., regression trees – Breiman et al., 1984), ‘classical’ symbolic learning methods cannot be used for such numeric problems (unless the domain of the numeric target concept can be abstracted into discrete, qualitative subranges without loss of relevant information). In particular, plausible explanation methods capable of utilizing qualitative domain knowledge, like those mentioned above, are not applicable to such domains; they assume discrete target concepts, and it is not clear how the qualitative background knowledge should be related to the precise numeric information in the data.

The topic of this paper is a new learning model (and an implemented system) that can learn numeric target concepts while taking maximum advantage of available qualitative domain knowledge, given that the problem and the target concepts satisfy some basic assumptions. The approach consists essentially in using a *symbolic learner* to partition the space for an *instance-based, numeric method* that is used to predict precise values of the target variables. The symbolic learner produces *plausible explanations* for discrete subconcepts; the explanations (and the extracted rules) are based both on qualitative background knowledge and on empirical information from the training data. The underlying instance-based method stores

the examples in several *independent instance spaces* and uses the learned symbolic rules to decide which instance space is relevant to a given example. Values for new examples are then predicted by numeric interpolation in those instance spaces that are classified as relevant by the associated symbolic rules.

The motivation for this research was a practical and complex problem in the domain of tonal music, namely, learning to apply *expressive interpretation* to a given piece of music, i.e., to dynamically vary tempo and dynamics in order to produce a musically satisfying performance. The target concepts in this problem are necessarily numeric (exactly how much variation should be applied to a given note), and there is some natural domain knowledge that is relevant to the task. The domain knowledge comes from music theory and is inherently qualitative and incomplete, but describable in explicit form.

The model has been implemented in a learning system named IBL-Smart (for reasons that will become obvious soon). We will first present the general learning model, then describe its realization in the system IBL-Smart, and illustrate its applicability with a description of our particular musical application and some experimental results. Our approach was strongly inspired by ideas presented in (DeJong, 1989), and the last section will relate our system to that and other work.

2 The General Model

2.1 Statement of the learning problem

This paper deals with supervised learning of numeric target concepts. More precisely, the class of learning problems we are interested in can be defined as follows:

Given: a set E of training examples, described in terms of a set of operational predicates P , where we distinguish symbolic predicates PS and numeric predicates (attributes) PN . Thus, $PS \cup PN = P$. Also attached to each example $e \in E$ is a numeric attribute $T(e, v)$ (the *target attribute*) with known value v . (This replaces the *classification* in symbolic supervised concept learning.) As v is a function of (the description of) the instance, we will also write $v = T(e)$. Note that so far, there are no negative instances in this scenario.

Find: a set of general rules that predict, for any given object o described by predicates $\in P$, a numeric value $v = T(o)$, based on the description of o .

(As in symbolic concept learning, we might require these rules to be *complete* (predict a value for every example) and *correct* (predict the correct value for each example) with respect to the training data (cf. Michalski, 1983). However, this may not be 100% desirable or feasible in every application domain.)

In addition, there may be some domain-specific *background knowledge* (BK) relating the target concept $T(X, V)$ (or some abstractions of T – see below) to some of the operational predicates P in specific ways, possibly via some intermediate non-operational predicates. This knowledge might be in the form of rules, as in standard EBL domain theories (Mitchell et al., 1986) or in the form of qualitative knowledge items as in (Widmer, 1993). The knowledge need not be correct or complete, nor need it be quantitative and precise. An additional constraint then is to find solutions (rules) that conform as closely as possible to BK while also consistently describing the training data E .

The learning model we are going to introduce in the next section includes a symbolic learning component that can utilize qualitative background knowledge for generating plausible explanations. For this method to be applicable, we need to make the following

Assumptions:

- 1) We assume that there are some discrete, qualitative sub-concepts $T_i(X)$ of the target concept $T(X, V)$ that can naturally be distinguished, where a sub-concept is defined by a more or less clearly distinguished subrange of the function value V .
- 2) We further assume that it is these discrete sub-concepts that are related to operational predicates P by the available background knowledge BK .
- 3) Finally, we assume that examples of the discrete sub-concepts T_i can be distinguished using the operational predicates P .

For example, in our energy demand prediction task, such subconcepts might be *extremely_low(Demand)* or *higher_than_capacity(Demand)*; in the musical domain described below, there are natural qualitative subconcepts such as *crescendo(Note)* and *diminuendo(Note)* (increase or decrease, respectively, in loudness relative to the current level) or *accelerando(Note)* and *ritardando(Note)* (increase or decrease in tempo).¹⁾

The motivation for this assumption is that these discrete, qualitative, symbolic sub-concepts will be the target concepts for the symbolic

¹⁾ The boundaries between these subconcepts will sometimes have to be defined somewhat arbitrarily. This is not necessarily a problem, as the results of the symbolic learning component are not used for classification, but only to find appropriate sets of instances for comparison. Section 2.2 will make that clearer.

learning component. Each of the original training instances will be assigned to one of the sub-concepts T_i , depending on its value $v = T(e)$, and the symbolic learner will learn general rules for each sub-concept. Note that in this way we also introduce negative instances for each target concept T_i , namely, all examples assigned to some T_j where $j \neq i$.

2.2 The learning model

Returning to our general learning problem, one way to approach it would be to simply do instance-based learning in the entire description space spanned by all the available attributes P , symbolic and numeric. That is, training instances would be stored along with their complete descriptions, and the value $v = T(o)$ for some new object o would be predicted by some nearest neighbor method in the space of stored instances, possibly with some numeric interpolation. There are several problems with this approach. First, it is not always clear how to devise a similarity metric that combines symbolic and numeric attributes in a meaningful way, especially when the attributes are of various types and inhomogeneous with respect to domain size etc. Second, the only way to integrate available qualitative background knowledge into the learning process is via the similarity metrics. This may not be the most natural way to express one's domain knowledge. Moreover, instance-based approaches suffer from the problem that they do not produce comprehensible concept descriptions. On the other hand, it is clear that some kind of instance-based interpolation component is needed in such domains, as the task is to predict numeric values from continuous domains, which is impossible with discrete, symbolic concept descriptions.

The model we are proposing here consists of two components: a *symbolic learning component* that learns to distinguish different types of situations and can utilize all the available domain knowledge, and an *instance-based component* which stores the instances with their precise numeric attribute values and can predict the target value for some new object by numeric interpolation over known instances. The connection between these two components is as follows: each rule (conjunctive hypothesis) learned by the symbolic learning component describes a subset of the instances; these are assumed to represent one particular subtype of the concept to be learned. All the instances covered by a rule are given to the instance-based learner to be stored together in a separate instance space. Predicting the target value for some new object then involves matching the object against the symbolic rules and using only those numeric instance spaces (interpolation tables) for prediction whose associated rules are satisfied by the object. In this way, the system learns several distinct instance spaces where different laws and regularities may apply. In fact, different instance spaces may contain examples with conflicting values.

More precisely, the target concepts for the *symbolic learning component* are the discrete, qualitative sub-concepts T_i mentioned in section 2.1. The symbolic learner learns general conditions that characterize or discriminate between these discrete classes. These conditions may refer to both symbolic and numeric predicates. The symbolic learner tries to use all the available qualitative background knowledge. The result produced by this component is a set of general rules that group the examples into clusters by assigning them to different sub-classes of the target concept.

The *numeric, instance-based component* takes the original training instances E as clustered by the symbolic learner, and creates a separate instance space from each cluster. Instances are stored with all their numeric attributes and with their precise numeric target values. The dimensions of such an instance space are thus defined by the numeric attributes PN . For some new object o , the target value $v = T(o)$ can then be predicted by selecting the appropriate instance space (by using the generated symbolic rules as filters), and applying some numeric interpolation method over the stored instances.

Several comments seem to be in order here: First, we assume that the symbolic learning component may refer to predicates from both PS and PN for its hypotheses. It is not realistic (nor necessary) to expect that the discrete sub-concepts T_i can always be distinguished by reference to symbolic predicates only. Second, we do assume that after clustering the examples according to sub-concepts (and sub-sub-concepts, if these are disjunctive), interpolation over the *numeric* attributes in the resulting instance spaces is sufficient to predict sensible target values. In other words, we assume that the rules learned for the sub-concepts T_i contain all the relevant *symbolic* information. The dimensions of the instance spaces are only attributes from PN . Any other solution would require some non-standard interpolation scheme to arrive at numeric prediction values. If additional domain knowledge about attribute relevance is available, the number of numeric dimensions may still be reduced, or some specialized similarity measures may be used for interpolation.

3 Realization of the Model: IBL-Smart

The general method has been implemented in a system named IBL-Smart and has been tested in the context of a complex musical problem. In accordance with the model, IBL-Smart consists of two components. The first of these — the symbolic learner — has been specifically designed to be able to use qualitative domain knowledge.

3.1 The symbolic learning component

The symbolic learner in IBL-Smart is a multiple-instance plausible explanation system based on the search algorithm of ML-Smart (Bergadano & Giordana, 1988). It performs top-down discrimination, integrating and interleaving deductive and inductive operationalization steps. The basics of the search are described below (section 3.1.1). For IBL-Smart, we have extended ML-Smart's discrimination algorithm to also use qualitative background knowledge in the form of *general dependency statements* and *directed qualitative dependency relations*. This is described in section 3.1.2.

3.1.1 The basic search algorithm

The learner starts with a nonoperational definition of the target concept (some discrete sub-concept T_i) and performs stepwise operationalization (specialization) by growing a heuristic best-first search tree. Each node/partial hypothesis in the search tree is accompanied by its extension, i.e., the positive and negative examples covered by the operational part of the expres-

sion. This makes it possible to use coverage measures as part of the search heuristic.

As in ML-Smart, each step in the search is either

- (1) a *deductive application of a rule* from the domain theory — replacing a non-operational literal by its sufficient conditions as defined by the rule;
- (2) an *inductive generalization step* — dropping a predicate when the node covers too few positive instances and thus the hypothesis seems too restricted; or
- (3) an *inductive specialization step* — adding some predicate to the operational part of the hypothesis in order to exclude some negative instances.

Deductive operationalization steps (1) are preferred. Inductive specialization (3) is done when deductive operationalization is not possible (e.g., when no rule is available or applicable to the examples). Inductive generalization (2) is attempted whenever the current node covers too few positive instances (according to some threshold) and thus the hypothesis seems too restrictive. The system then looks for a condition that, if dropped, would increase the number of instances covered by the hypothesis. All in all, the search algorithm integrates deduction and induction in a fine-grained manner.

The search is guided by a heuristic measure (*evaluation function*) H that measures the relative 'goodness' of nodes. The heuristic decides both which node is to be expanded next, and how. Among other things (see below), it takes into account the coverage of the expression, i.e., the ratio positive/negative instances covered by the node, and also the absolute number of positive instances covered.

The discrimination algorithm has been extended to utilize also numeric attributes in discrimination steps. For numeric attributes, the system looks for a binary split point that best discriminates between positive and negative instances, as it is done in some decision tree learners (e.g., Cestnik et al., 1987; Fayyad & Irani, 1992). The general evaluation function H of the search algorithm is used to determine what is the best split.

3.1.2 Using qualitative background knowledge

The search algorithm as described above corresponds closely to the original ML-Smart method as presented in (Bergadano & Giordana, 1988). In our system IBL-Smart, the algorithm has been extended so as to also utilize *qualitative* background knowledge, where available. IBL-Smart domain theories may contain two types of qualitative knowledge items:

- (1) *General dependency statements* of the form $\text{depends_on}(Q, Ps)$ simply state that some predicate Q may be operationalized by using a set of specified predicates Ps . This type of general knowledge items has already been proposed in (Bergadano, Giordana and Ponsero, 1989). In IBL-Smart, such statements tell the search algorithm to use an entire set of predicates in one operationalization step: successors of a node are created for *all possible combinations* of values for the predicates Ps occurring in some positive instances covered by the node.

Such dependency statements are similar, but not identical, to *determinations* (Russell, 1987). They permit IBL-Smart to perform strictly constrained forms of look-ahead, and thus help overcome blindness

effects that would arise if the algorithm performed purely empirical step-wise specialization. For instance, they can be used to describe *relational clichés* as proposed in (Silverstein & Pazzani, 1991).

- (2) *Directed dependency statements* of the form $q+(A,B)$ can be paraphrased as "the values of A and B are positively proportionally related" or "high (or low) values of A tend to produce high (or low) values of B, all other things being equal". Negative dependency ($q-(A,B)$) is defined analogously. Such statements are, of course, restricted to functional predicates (or *attributes*) that assign values to objects. They were already used in (Widmer, 1991) and are similar to Michalski's *M-descriptors* (Michalski, 1983). The notation was borrowed from Forbus' *qualitative proportionalities* (Forbus, 1984).

In the search algorithm of IBL-Smart, directed dependencies are used like general dependencies (create successors for all possible value combinations), and the additional knowledge about the *direction* of influence is used in the *search heuristic H*: when evaluating some operationalization based on a $q+$ or $q-$ relation, the heuristic also rates the degree to which the particular values involved match the direction of the dependency (which is assumed to be linear). Knowing that $q+(A,B)$ and operationalizing condition $B(.) = b$, an operationalization $B(.) = b$ because $A(.) = a$ for specific values b and a will be regarded the more plausible the more the relative positions of b and a in their respective domains agree: $B(.) = \text{high}$ because $A(.) = \text{high}$ is rated as more plausible than $B(.) = \text{high}$ because $A(.) = \text{low}$ (see also Widmer, 1993.). Hy-

potheses constructed by IBL-Smart will tend to include those attributes that most closely approximate such linear constraints between the data and the background knowledge.

Note that, as with strict deductive rules, such qualitative dependency statements need not necessarily be entirely correct in order to have a positive impact on the search. If a dependency statement is correct, it will lead to faster convergence; if it is too general (the given predicates are not sufficient to completely discriminate between positive and negative instances), subsequent empirical discrimination steps will refine it. And if it is overly restrictive (some predicates are not necessary), this may be repaired by empirical generalization steps, where the predicates that are too restrictive are removed from the hypothesis to arrive at a more general partial concept description.

By taking into account both such inference-dependent plausibility measures and information about the numbers of positive and negative instances covered by a node, the search heuristic combines weak, imprecise background knowledge with empirical information from the training data, producing hypotheses that tend to correspond to the background knowledge as much as the data permit and overriding the background knowledge if the data are in conflict with the knowledge.

3.2 The numeric instance-based component

The result of this learning step is a concept hypothesis for a discrete, qualitative sub-concept in the form of a DNF expression, where each conjunct describes one particular subtype of the sub-concept. The instance-based learner now collects all the training instances covered by a particular conjunct and builds an instance store

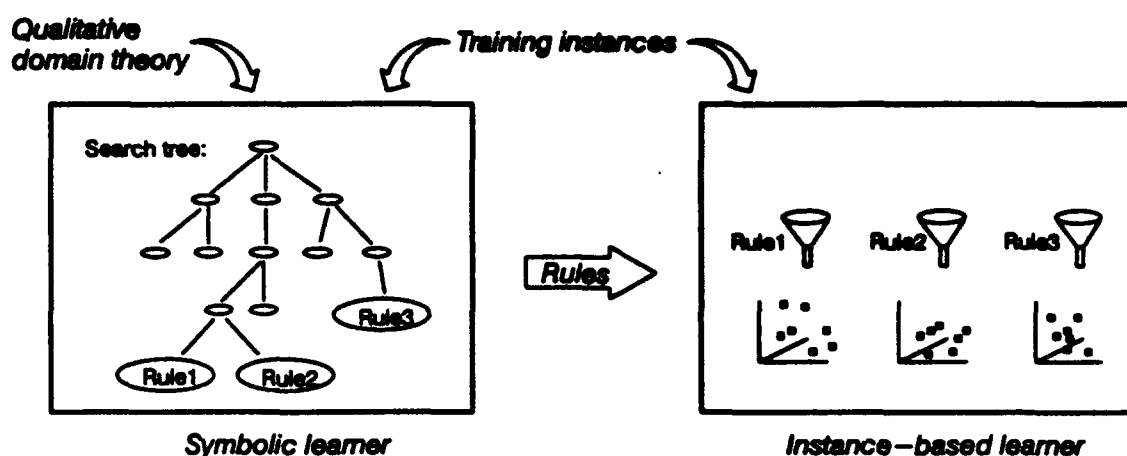


Fig. 1: Sketch of IBL-Smart

in the form of an *interpolation table*, using these examples. In the absence of knowledge about the relevance of the numeric attributes (PN) to the target value, the *dimensions* of the interpolation table are chosen to be all the numeric attributes ($\subseteq PN$) shared by the selected instances (not all instances may have defined values for all attributes), and the output dimension is the value of the target variable $V = T(X)$.

When given a new instance for which to predict the value of the target variable, the system matches the instance against all learned rules, retrieves those instance spaces whose associated rules are matched, and computes a value for the instance's target value by interpolation in each of the retrieved spaces. If the instance matches more than one rule, and thus target values are computed in several spaces, the target values are simply averaged. Lacking more specific knowledge about the relationships between the various numeric parameters, we use the Euclidean distance as the similarity measure and perform linear interpolation. Figure 1 summarizes the basic structure of IBL-Smart.

4 An Application of IBL-Smart: Learning Expressive Interpretation

IBL-Smart has been applied to a complex problem from the domain of tonal music, namely, expressive performance or interpretation of written music. By this we understand the variations in tempo and loudness that a performer applies (consciously or unconsciously) to the notes of a piece during performance. When played exactly as written, most pieces of tonal music would sound utterly mechanical and lifeless.

There are basically three dimensions to expressive performance: variations in tempo ("*rubato*"), in loudness ("*dynamics*") and in the duration of notes as actually played, as opposed to the notated length ("*articulation*").

In this presentation, we will restrict ourselves to the dimension of *dynamics*. As mentioned in the introduction, this concept is inherently numeric, as the task is to decide not just whether or not to play some note louder or softer, but exactly by *how much*. Nevertheless, there are two discrete, qualitative sub-concepts that can naturally be distinguished: *crescendo*(Note) and *diminuen-*

do(Note) – whether a note is to be played louder or softer, respectively, than some standard level. These are the target concepts for the plausible explanation component. The precise amounts by which the loudness is to be varied are numeric multiplication factors that are to be learned by the instance-based component.²⁾

Training instances are derived from actual performances of piano pieces recorded on an electronic piano via a MIDI interface. At the moment, we are restricting ourselves to single line melodies (with additional information about the underlying harmonic structure of the piece). That is, the input is a sequence of notes, described in terms of various predicates and accompanied by explicit information about the degree of crescendo or diminuendo that was applied to it by the performer. Each note of a played piece is a training instance.

The *description language* consists of predicates that describe various features of a note and structural features of its surroundings. There are currently 41 operational predicates, of which 21 are symbolic (like *followed_by_rest*(Note)) and 20 are numeric (like *duration*(Note,X)). Some of

these predicates are computed by a pre-processing component which performs a music-theoretic analysis of the given piece in terms of some relevant musical structures (e.g., phrases and various types of ‘processes’ such as linear melodic lines (ascending or descending), rhythmic patterns, etc.). Many numeric attributes then describe the relative position of a note in a phrase or in a ‘process’. Note that the number of attributes defined for a given note varies: some notes occur in many patterns, others only in some. So not all numeric attributes are defined for every note.

The background knowledge for this problem is mainly in the form of directed and undirected dependency statements. The domain theory is a hierarchy of such dependency statements and some crisp rules. The top level of the theory relates the phenomenon of loudness variations to some abstract musical notions by a set of dependencies like

```
depends_on( crescendo(Note,X),
            [ salience(Note,Y)]).
depends_on( crescendo(Note,X),
            [ goal_directedness(Note,Y)]).
depends_on( crescendo(Note,X),
            [ closure(Note,Y)]).
```

The first of these can be paraphrased as

“Whether crescendo should be applied to a note (and if so, the exact amount X) depends, among other things, on the structural importance (salience) Y of the note.”

and analogously for the other ones.

The abstract notions *salience*, *goal_directedness*, and *closure* are then again related to lower-level musical effects, all the way down to some surface features of training instances, for example:

2)

A clarifying remark to readers who feel that we are trivializing the artistic phenomenon of expressive musical performance by claiming that a computer program can easily learn to replicate such behaviour, or that these phenomena can be explained by some simple domain theory: We are not talking here about the highly artistic details in variation that distinguish a great pianist or other performer. We are convinced, however (and there is much support for this hypothesis from various areas of musicology), that expressive performance does have a large ‘rational’ component, in that one of its purposes is to convey an understanding of musical structure to a listener. It is this rational part for which we can find partial plausible explanations and which we can expect a computer to learn, provided it is equipped with the necessary musical knowledge and a suitable vocabulary.



Fig. 2: Beginnings of three little minuets by J.S.Bach

$q+(\text{metrical_strength}(\text{Note},X),$
 $\text{stability}(\text{Note},Y))$ and
 $q+(\text{harmonic_stability}(\text{Note},X),$
 $\text{stability}(\text{Note},Y))$

"The degree of stability Y of a note is positively proportionally related (among other things) to the metrical strength X of the note" etc.

where *metrical_strength* is a numeric and *harmonic_stability* is a symbolic attribute (with a discrete, ordered domain of qualitative values). Both are defined as operational.

Given this domain theory and some played pieces, the plausible explanation component learns mixed symbolic/numeric rules that discriminate various types of situations where a crescendo or a diminuendo occurs. These rules are sets (disjunctions) of conjunctive conditions; each conjunct describes a particular class of crescendo/diminuendo situations. For each conjunct, a numeric interpolation table (instance space) is created which contains all the instances covered by the conjunct. The set of all numeric attributes shared by all the instances

covered by a conjunct defines the dimensions of the respective interpolation table.

5 An Experiment

Several experiments with comparatively simple piano pieces have been performed. In one experiment, we chose three well-known minuets from J.S.Bach's *Notenbüchlein für Anna Magdalena Bach* as training and test pieces. The beginnings of the three minuets are shown in Figure 2. All three pieces consist of two parts. The second part of each piece was used for training: they were played on an electronic piano by the author, and recorded through a MIDI interface. After learning, the system was tested on the first parts of the same pieces. In this way, we combined some variation in the training data (three different pieces) with some uniformity in style (three pieces from the same period and with similar characteristics; test data from the same pieces as training data, though different).

The training input consisted in 212 examples (notes), of which 79 were examples of cresc-

do, and 120 were examples of diminuendo (the rest were played in a neutral way). The system learned 14 rules (conjuncts) and, correspondingly, 14 interpolation tables characterizing crescendo situations, and 15 rules for diminuendo. Quite a number of instances were covered by more than one rule. For illustration, here is a simple rule for crescendo:

```
crescendo(Note,X) :-
    metrical_strength(Note,S),
    S > 4.0,
    harmony_stability(Note,high),
    previous_interval(Note,I1),
    direction(I1,up),
    next_interval(Note,I2),
    direction(I2,down).
```

"Apply some crescendo to the current note if the metrical strength of the note is > 4 and the underlying harmony is stable and the direction of the melody from the previous to the current note is up and the direction of the melody from the current note to the next is down"

The quality of the learning results is not easy to measure, as there is no precise criterion to decide whether some performance is right or wrong. Judging the correctness is a matter of listening. Unfortunately, we cannot attach a recording to this paper so that the reader can appreciate the results. Instead, Figure 3 depicts a part of one of the training pieces (the second part of the first minuet in G major as played by the author), and also shows the performance created by the system for a test piece (the first part of the same minuet) after learning. The figures plot the relative loudness with which the individual notes were played. A level of 1.0 would be neutral, values above 1.0 represent crescendo (increased loudness), values below 1.0 diminuendo.

The reader familiar with standard music notation may appreciate that there are strong similarities in the way similar types of phrases are played by the human teacher and the learner. (Note, for instance, the crescendo in lines rising by stepwise motion, and the decrescendo patterns in measures with three quarter notes). Generally, the results were very good, given the limited amount of training data and the surface differences between training and test pieces. Readers not familiar with music notation will have to take our word for it. We are planning experiments with other, non-musical domains where the results will be more easily interpretable and testable.

In a comparative experiment, we also tested a system restricted to learning only in an instance-based way, that is, with interpolation tables, but without the symbolic explanation component. This learner used *all* the available attributes, both numeric and symbolic. The following distance metric was used: all numerical attributes were scaled between 0 and 1, and for symbolic attributes, the distance was defined to be 0 in the case of a match and 1 otherwise. As not all training instances share all numeric dimensions, the system learned as many interpolation tables as there were combinations of numeric attributes occurring in the training data (18 for crescendo, 12 for decrescendo). The results on the same data were considerably worse. The learner did not distinguish as well between different types of situations, and the results are rather blurred, as can also be seen from Figure 4, which shows the same test piece as played by the second system.

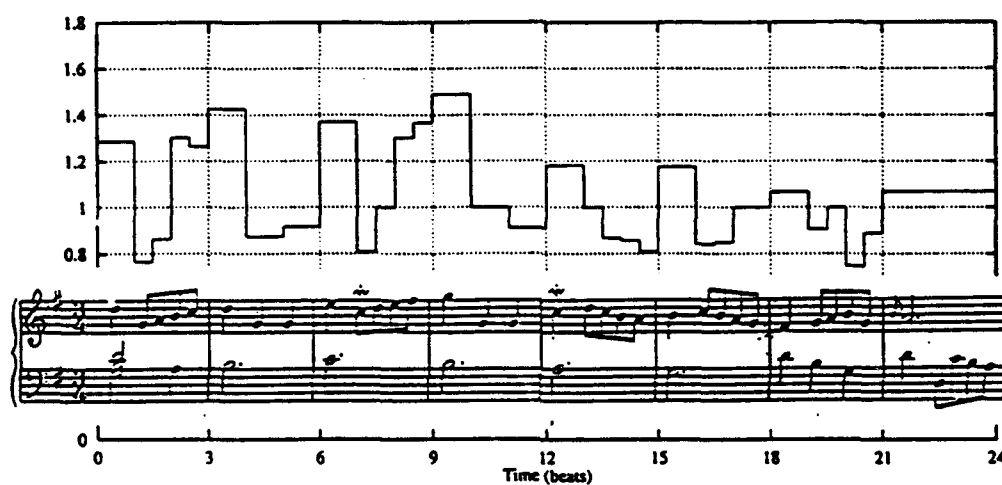
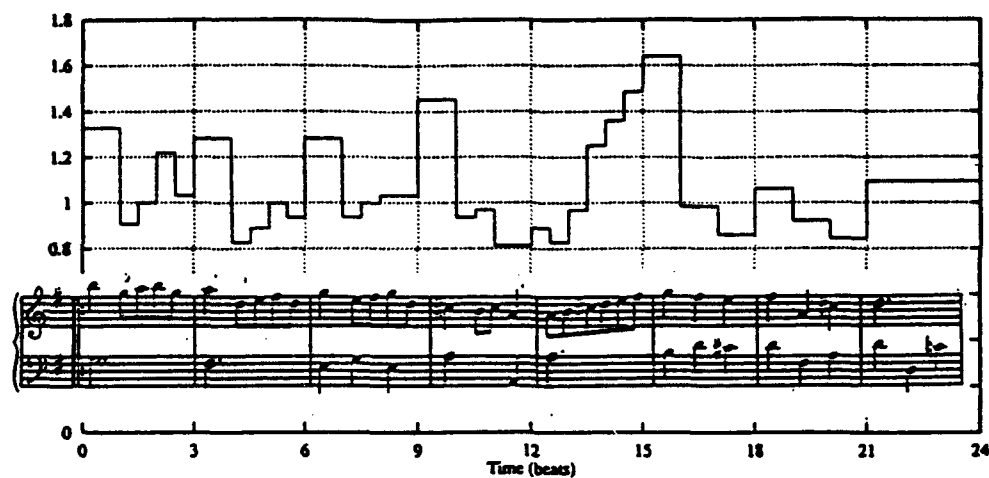


Fig. 3: Parts of a training piece as played by teacher (top) and test piece as played by learner after learning (bottom)

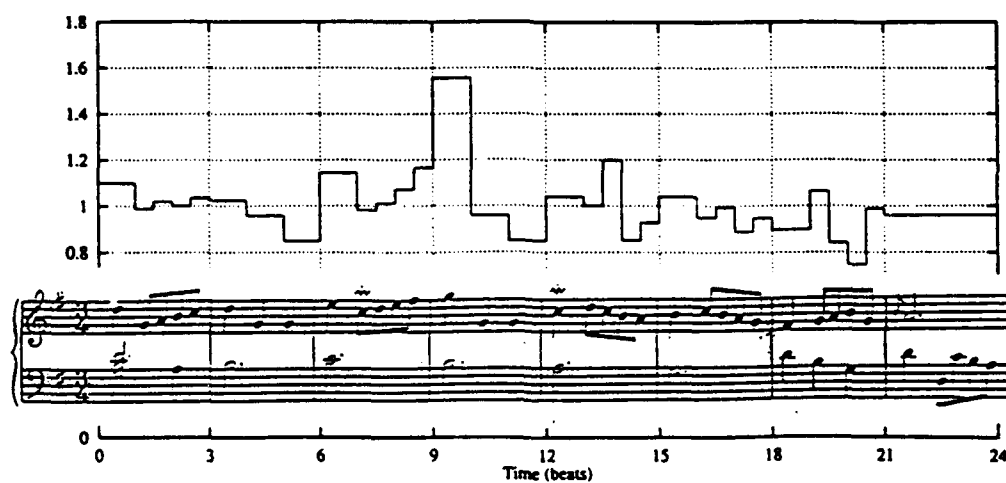


Fig. 4: Part of test piece as played after instance-based learning only

6 Discussion, Related Work, and Related Matters

First, let us briefly recapitulate the main characteristics of the learning model: (1) The model can learn precise numeric concepts via an instance-based method while using available qualitative background knowledge through a symbolic learning component. (2) The symbolic learning component defines and separates different independent regions in numeric instance space where different regularities may apply. This allows the instance-based learner to build specialized instance stores, which may yield very specific prediction behaviour. And (3), as a side effect, learning rules for discrete sub-concepts clusters the examples around meaningful abstractions, which may be useful for other tasks.

The definition of abstract sub-concepts T_i introduces a natural distinction between symbolic and numeric learning, and also produces negative instances for the symbolic learner. That the background knowledge is used only by the symbolic component seems natural, given that it is qualitative and thus may explain abstract, symbolic concepts (at best), but certainly not precise numeric values and relationships.³⁾ Of course, this does not preclude the use of additional knowledge to guide or constrain numeric learning in the instance-based component.

³⁾ For instance, in music, we may be able to explain why a performer applied some crescendo at a certain point (for instance, in order to stress a musically important event), but we can never explain why she chose exactly that *precise degree* of crescendo. A system can only record these precise degrees and try to replicate the same behaviour in similar situations. What is similar is determined by the rules learned by the symbolic component.

It should be remembered that this is a general learning model: the system presented here — IBL-Smart — is just one particular incarnation of a more general approach. We have found it convenient to use a best-first search algorithm like the ML-Smart learner as the basis for our plausible explanation component, as it explicitly constructs a search tree and allows us to integrate various sources of knowledge into the learning process via the search heuristic (evaluation function). However, with appropriate modifications and extensions, other symbolic learners capable of utilizing incomplete and inconsistent knowledge — for instance, FOCL (Pazzani & Kibler, 1992) — might be used just as well in this framework.

Similarly, more elaborate strategies could be used in the instance-based component. (Aha et al., 1991) have described a number of instance-based learning methods that could be applied within a framework such as ours. Also, available domain knowledge about the relative degree of relevance of numeric attributes or about the domains and typical values of numeric variables could be used to devise more sophisticated similarity metrics, tailored to the particular application.

With respect to related work, we acknowledge the important influence on this project by some of the ideas expressed in (DeJong, 1989). DeJong had presented a system that combined a very weak notion of plausible inference over single cases with numeric variables. Our approach departs from his, among other things, in the variety of types of background knowledge and in the use of a heuristically guided, search-based, multi-instance explanation algorithm that allows much more control over the learning process. Not only does this search introduce a strong notion of *empirical plausibility* by taking

into account the distribution of instances; the use of an explicit search heuristic also makes it possible to exploit the qualitative knowledge contained in *qualitative dependencies* ($q+$, $q-$) to compute the relative plausibility of arguments. The best-first search is very likely to find explanations that are most plausible overall (both with respect to the knowledge and the data). DeJong's system, on the other hand, simply assumed that the syntactically simplest explanation was also the most plausible one.

As an additional advantage of this multi-instance explanation approach, we note also that there is a natural way to deal with certain types of *noise* in the training data. The evaluation function of the search algorithm incorporates two thresholds: it accepts only nodes (conjunctions) that cover some minimum number of positive instances, and the termination criterion allows the search to halt when a certain percentage ($< 100\%$) of positive instances are covered.

Thus, the system can ignore rare instances that look like exceptions, but are really the result of noise. By varying these thresholds, the system can be tuned to the characteristics of different application domains.

In fact, the musical experiments described in the previous section were characterized very strongly by noise in the data, originating from the author's imperfect piano technique, from the imprecise boundaries between the abstract sub-concepts crescendo and diminuendo, and from imprecision inherent in the domain itself (there are simply no 100% laws as to how some passage must and will be played; variation will invariably happen). The system concentrated on learning typical variations.

Of course (and this is also implied by the name), our system also owes a lot to the work on integrated deductive-inductive learning in ML-

Smart (Bergadano & Giordana, 1988). We have extended the ML-Smart algorithm to also utilize background knowledge in the form of directed dependency statements (which are a very natural kind of knowledge in many domains).

With respect to the system described in (Widmer, 1991; 1993), which also constructs plausible explanations of individual training instances on the basis of qualitative background knowledge, we note that explaining multiple instances at a time adds a strong empirical justification to plausible explanations. The price is non-incrementality. However, it is likely that, using techniques described in (Widmer, 1989), IBL-Smart can be made to learn incrementally without losing too much in effectiveness.

Acknowledgments

I would like to thank Johannes Fürnkranz for helpful comments on this paper. Thanks also to the anonymous reviewers for very precise and stimulating comments. This research is sponsored in part by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant P8756-TEC. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry for Science and Research.

References

- Aha, D.W., Kibler, D., and Albert, M.K. (1991). Instance-Based Learning Algorithms. *Machine Learning* 6(1), pp. 37-66.
- Bergadano, F. and Giordana, A. (1988). A Knowledge Intensive Approach to Concept Induction. In *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, pp. 305-317.
- Bergadano, F., Giordana, A., and Ponsero, S. (1989). Deduction in Top-Down Inductive Learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, N.Y., pp. 23-25.

- Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.
- Cestnik, B., Kononenko, I., and Bratko, I. (1987). ASSISTANT 86: A Knowledge Elicitation Tools for Sophisticated Users. In I. Bratko & N. Lavrac (Eds.), *Progress in Machine Learning*. Wilmslow, U.K.: Sigma Press.
- Collins, A. and Michalski, R.S. (1989). The Logic of Plausible Reasoning: A Core Theory. *Cognitive Science* 13(1), pp. 1-49.
- DeJong, G. (1989). Explanation-Based Learning with Plausible Inferencing. In *Proceedings of the Fourth European Working Session on Learning (EWSL-89)*, Montpellier, France, pp. 1-10.
- Fayyad, U. and Irani, K. (1992). On the Handling of Continuous-Valued Attributes in Decision Tree Generation. *Machine Learning* 8(1), pp. 87-102.
- Forbus, K.D. (1984). Qualitative Process Theory. *Artificial Intelligence* 24(1-3), pp. 85-169.
- Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach, vol. I*. Palo Alto, CA: Tioga.
- Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. (1986). Explanation-Based Generalization: A Unifying View. *Machine Learning* 1(1), pp. 47-80.
- Pazzani, M. and Kibler, D. (1992). The Utility of Knowledge in Inductive Learning. *Machine Learning* 9(1), pp. 57-94.
- Russell, S.J. (1987). *Analogical and Inductive Reasoning*. Ph.D. thesis, Report STAN-CS-87-1150, Stanford University, Stanford, CA.
- Silverstein, G. and Pazzani, M.J. (1991). Relational Clichés: Constraining Constructive Induction During Relational Learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, Ill., pp. 203-207.
- Tecuci, G.D. (1991). Learning as Understanding the External World. In *Proceedings of the First International Workshop on Multistrategy Learning*, Harper's Ferry, W.VA, pp. 49-64.
- Tecuci, G.D. and Michalski, R.S. (1991). A Method for Multistrategy Task-adaptive Learning Based on Plausible Justifications. In *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, Ill., pp. 549-553.
- Widmer, G. (1989). An Incremental Version of Bergadano & Giordana's Integrated Learning Strategy. In *Proceedings of the Fourth European Working Session on Learning (EWSL-89)*, Montpellier, France, pp. 227-238.
- Widmer, G. (1991). Learning by Plausible Reasoning and its Application to a Complex Musical Problem. In *Proceedings of the First International Workshop on Multistrategy Learning (MSL-91)*, Harper's Ferry, W.VA.
- Widmer, G. (1993). Learning with a Qualitative Domain Theory by Means of Plausible Explanations. In R.S. Michalski and G. Tecuci, eds., *Machine Learning: A Multistrategy Approach, vol. IV*. Los Altos, CA: Morgan Kaufmann. (in press)

***k*-DT: A Multi-Tree Learning Method**

David Heath, Simon Kasif, and Steven Salzberg

Department of Computer Science

The Johns Hopkins University

Baltimore, MD 21218

(410) 516-8296

lastname@cs.jhu.edu

Abstract

This paper introduces a technique for using the randomized nature of some learning algorithms to increase their accuracy. Our method is to generate multiple classifiers and combine them with a majority voting scheme. The purpose of this technique is to overcome small errors that appear in individual classifiers. We have tested our idea on a type of randomized decision tree with real data, and found that it consistently improves the accuracy over that of average trees. We have also shown that this technique outperforms some other methods that attempt to improve accuracy by using randomization in a different way.

Quinlan, 1987) and incremental versions of the algorithms (Utgoff, 1989). Many of these refinements have been designed to produce better decision trees; i.e., trees that were either more accurate classifiers, or smaller trees, or both.

The main goal of our research is to produce classifiers that provide the most accurate model possible for a set of data. To achieve our goal, we have combined a standard method for classification - decision trees - with two other ideas. The first idea is randomization, which in this context allows us to generate many different trees for the same task. The second idea is majority voting, which is used, *e.g.*, by *k*-nearest-neighbor methods to decide on a classification. Here we use a majority vote of *k* decision trees to classify examples.

1 Introduction

Decision trees have been used successfully for many different decision making and classification tasks. A number of standard techniques have been developed in the machine learning community, most notably Quinlan's ID3 algorithm (1986) and Breiman *et al.*'s CART algorithm (1984). Since the introduction of these algorithms, numerous variations and improvements have been put forward, including new pruning strategies (*e.g.*,

2 Randomization in Learning Algorithms

In a previous work, [Heath, 1992], we introduced the SADTlearning algorithm (described below). In that work, we explored the generation of decision trees comprised of tests that are linear inequalities over the attributes (*oblique* decision trees). This is a

generalization of standard decision tree techniques, in which each node of a tree is a test of a single attribute. We showed that, when generating oblique trees, finding even a single test that minimizes some goodness criteria is an NP-hard problem. We then turned to the optimization technique of simulated annealing to find *good* tests, which should generate good (i.e., small and accurate) trees.

Using simulated annealing in our learning algorithm introduces an element of randomness. Each time our SADTprogram is run, it generates different trees. This led us to explore methods of using this randomization to our advantage by generating many trees and using an additional criteria to choose the best tree. Our argument was that picking a good tree out of the many solutions produced by a randomized algorithm may be preferable to using an algorithm, even a very clever one, that only produces one solution.

In this paper, we explore another way of using randomization to advantage. As before, we use a single training set to generate a set of classifiers. Instead of choosing one representative tree, we attempt to combine the knowledge represented in each tree into a new, more accurate, classifier.

Specifically, we take a set of classifiers and combine their classifications by taking the plurality. In binary classification problems, this reduces to taking the majority. For example, if we have 5 trees, and 3 classify an example as "0," and the other two classify it as "1," then we predict the example belongs to class "0." When this technique is applied to decision trees, we call the resulting algorithm *k*-DT, in the spirit of *k*-NN, the *k*-nearest-neighbor algorithm.

2.1 The advantage of majority voting

The premise behind this idea is that any one tree may not capture the target concept completely accurately, but will approximate it with some error. This error differs from tree to tree. By using several trees and taking the majority, we hope to overcome this type of error. Consider, for example, a test example x with probability $p(x)$ of being correctly classified by a random two-category SADTtree. If we take the majority vote of k trees, the probability that x is correctly classified is

$$maj(k, x) = \sum_{j > k/2}^{j \leq k} p(x)^j (1 - p(x))^{k-j} \binom{k}{j}$$

In this equation, j represents the number of trees that correctly classify example x . We require that it be more than half of the k trees, thus the restrictions on the sum. $p(x)^j$ represents the probability of j trees getting the example correct; $(1 - p(x))^{k-j}$ is the probability that the remaining trees get it wrong. $\binom{k}{j}$ simply counts the number of possible ways k trees could divide into two sets of trees, one of size j . Figure 1 shows how $maj(k, x)$ varies with $p(x)$ when different numbers of trees are used for the majority. Note that for example x , taking the majority vote increases the probability of getting a correct classification if $p(x) > 0.5$, but decreases it if $p(x) < 0.5$. Let X_1 be the set of examples in the test set for which $p(x) < 0.5$, and X_2 be those for which $p(x) > 0.5$. If $x \in X_1$, it is to our advantage to use the classifiers directly. If, on the other hand, $x \in X_2$, taking the majority will increase the probability that we will classify x correctly. For any given test set, there will likely be points in both cases. Obviously, we cannot tell, given a particular example, whether it belongs to X_1 or X_2 unless we know its clas-

sification. However, it is our experience that often the benefit we get by increasing the likelihood of a correct classification for those examples in X_2 outweighs the loss in accuracy we get on the examples in X_1 .

Contrary to intuition, simply increasing the number of trees participating in a majority voting scheme does not necessarily increase the expected accuracy of the classifier.

At first glance, it may appear that the more trees that are used, the higher will be the resulting accuracy. However, this is not necessarily true. An implication of this is that choosing the appropriate value for k may be a difficult problem.

We have already seen that for some examples (those with less than 50% probability of being correctly classified by the average tree), using a majority vote will lower the chances of a correct classification, and the more trees used, the lower the resulting accuracy will be. On the other hand, increasing the number of trees involved in the vote will increase the accuracy on those points likely to be classified correctly by the average tree. When we try using a majority voting scheme on a mixture of these two types of examples, we will get a mixed result. Consider two examples, e_1 and e_2 . If we generate many trees, on average e_1 is classified correctly 45% of the time, and e_2 is classified correctly 80% of the time. As shown in Figure 2, if we use a majority voting scheme, then e_1 will rarely be classified correctly, but e_2 will almost always be classified correctly. Figure 2 also shows the combined expected accuracy for the set $\{e_1, e_2\}$. If we generate a series of trees and use each one to classify the two examples, we expect their average accuracy to be 62.5%. If we use majority voting, we expect the accuracy to increase up to about 68% for nine trees. However, if we use more

than nine trees, the expected accuracy goes down, eventually converging to

For a set of examples X , where $p(x)$ is the probability of example x being correctly classified by an average tree, it is easy to show that the average accuracy without voting is

$$\frac{1}{|X|} \sum_{x \in X} p(x)$$

while the accuracy when an infinite number of trees are used in a majority computation is

$$\frac{|\{x \in X, p(x) > 0.5\}|}{|X|},$$

that is, the fraction of the examples which are more than likely classified correctly by the average tree. Between these two extremes, the overall accuracy may have dips and peaks.

In this paper, we try majority voting using different numbers of trees. We use these experiments to empirically choose a value for k which seems to work well in practice.

3 Related Work

k -DT is one of several different strategies for combining multiple classifiers. There are two common approaches to this problem. The first approach can be thought of as multi-level learning. A set of classifiers are trained. Their outputs are fed to another learning system, which learn an appropriate weighting scheme on the first-level classifiers, in the hopes of creating a more accurate classifier. Depending on the implementation, the two levels can be trained separately or simultaneously. Wolpert's [1992] stacked generalization technique and the hybrid technique developed by Zhang, *et al*, [1992]) are examples of separately trained systems. An

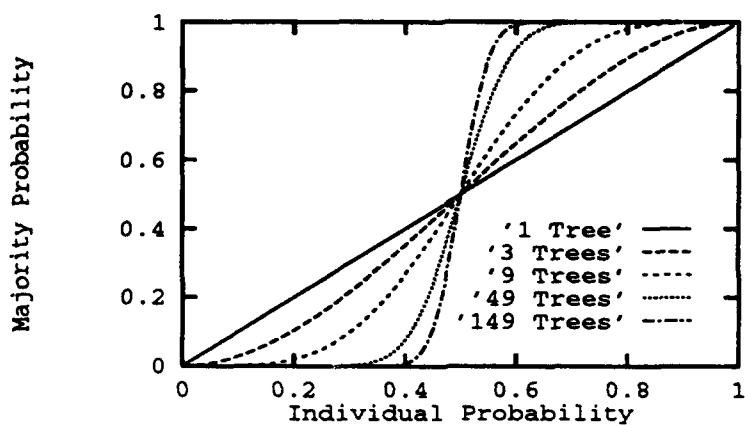


Figure 1: Majority classification probability vs. individual classification probability

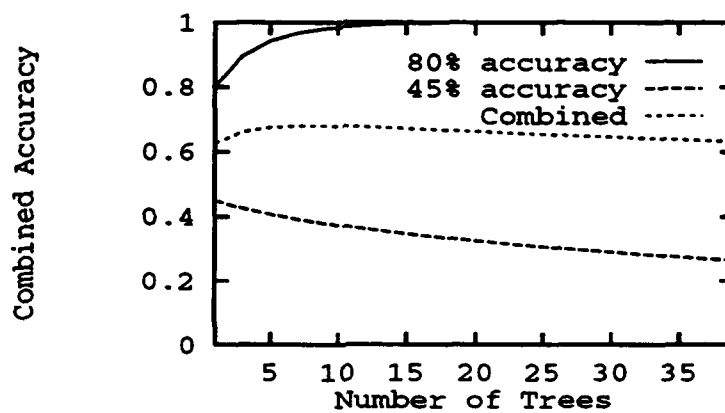


Figure 2: Effects of majority voting on mixed data sets

example of a simultaneously trained system is [Jacobs, *et al*, 1991], in which the second learning level learns how to assign training examples to the different components of the first level.

k -DT takes another approach. Only the first level is trained; the second level is a simple, easily understood, fixed network. Another system that shares this property is the cluster back propagation network of Lincoln *et al*, [1990]

4 The SADT Algorithm

Although the majority voting technique could be applied to any randomized classifier scheme, k -DT was first conceived of as a natural enhancement to our SADT algorithm. Accordingly, all of our experiments have been conducted on the SADT algorithm. To aid in the understanding of k -DT, we explain the workings of our SADT algorithm here.

The basic outline of the SADT algorithm is the same as that of most other decision tree algorithms. That is, we find a hyperplane to partition the training set and recurse on the two partitions. Here we describe the search for a good hyperplane.

In our implementation, d -dimensional hyperplanes are stored in the form $H(x) = h_{d+1} + \sum_{i=1}^d h_i x_i$, where $H = \{h_1, h_2, \dots, h_{d+1}\}$ is the hyperplane, $x = (x_1, x_2, \dots, x_d)$ is a point, and h_{d+1} represents the constant term. For example, in the plane the hyperplane is a line and is represented in the familiar $ax + by + c = 0$ form. Classification is done recursively. To classify an example, compare it to the current hyperplane (initially this is the root node). If an example p is at a non-leaf node labeled

$H(x)$, then we follow the left child if $H(p) \geq 0$; otherwise we descend to the right child.

The first step in our algorithm is to generate an initial hyperplane. The initial hyperplane we generate is always the same and is not tailored to the training set. We simply wanted to choose some hyperplane that was not parallel to any of the axes, so we used the hyperplane passing through the points where $x_i = 1$ and all other $x_j = 0$, for each dimension i . In particular, the initial hyperplane may be written in the above form as $h_i = 1$ for $1 \leq i \leq d$ and $h_{d+1} = -1$ since $H(x) = 0$ for each of these points. Thus in 3-D, we choose the hyperplane which passes through $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. Many other choices for the initial hyperplane would be equally good. Once the annealing begins, the hyperplane is immediately moved to a new position, so the location of the initial split is not important.

Next, the hyperplane is repeatedly perturbed. If we denote the current hyperplane by $H = \{h_1, h_2, \dots, h_{d+1}\}$, then the algorithm picks one of the h_i 's randomly and adds to it a uniformly chosen random variable in the range $[-0.5, 0.5]$. Using our goodness measure (described below), we compute the energy of the new hyperplane and the change in energy ΔE .

If ΔE is negative, then the energy has decreased and the new hyperplane becomes the current split. Otherwise, the energy has increased (or stayed the same) and the new hyperplane becomes the current split with probability $e^{-\Delta E/T}$ where T is the temperature of the system. The system starts out with a high temperature that is reduced slightly with each move. Note that when the change in energy is small relative to the temperature, the probability of accepting the

new hyperplane is close to 1, but that as the temperature becomes small, the probability of moving to a worse state approaches 0.

In order to decide when to stop perturbing the split, we keep track of the split that generated the lowest energy seen so far at the current node. If this minimum energy does not change for a large number of iterations (we used numbers between 3000 and 100,000 iterations in our experiments), then we stop making perturbations and use the split that generated the lowest energy. The recursive splitting continues until each node is pure; i.e., each leaf node contains only points of one category.

4.1 Goodness Criteria

SADT can work with any goodness criterion, and we have experimented with several criteria. For detailed discussions of these measures, see Heath [1992]. In this paper, we experiment with three of these criteria: Quinlan's [1986] *Information Gain*, and our own *Max Minority* and *Sum Minority*. We define max minority and sum minority as follows.

Consider a set of examples X , belonging to 2 classes, u and v . A hyperplane divides the set into two subsets X_1 and X_2 . For each subset, we find the class that appears least often. We say that these are the *minority* categories. If X_1 has few examples in its minority category C_1 , then it is relatively *pure*. We prefer splits that are pure; i.e., splits that generate small minorities. Let the number of examples in class u (class v) in X_1 be u_1 (v_1) and the number of examples in class u (class v) in X_2 be u_2 (v_2). To force SADT to generate a relatively pure split, we define the sum-minority error measure to be $\min(u_1, v_1) + \min(u_2, v_2)$, and the max-minority error measure to be

$$\max(\min(u_1, v_1), \min(u_2, v_2)).$$

5 Experiments

5.1 Classifying irises

For our first experiment, we ran k -DT on a real dataset that has been the subject of other machine learning studies. Fisher's iris data is a well known dataset (see Fisher [1936]), and many common learning techniques have been applied to it. Weiss and Kapouleas [1989] compared several learning algorithms on the iris data set, as well as some others. The data consists of 150 examples, 50 each of three different types of irises: *setosa*, *versicolor*, and *virginica*. Each example is described by numeric measurements of width and length of the petals and sepals.

We performed thirty-five 10-fold cross validation trials using SADT. In an x -fold cross-validation trial, we divide the dataset into x approximately equal sized subsets and perform x experiments. For each set s , we train the learning system on the union of the remaining $x - 1$ sets and test on set s . The results are averaged over these x runs.

SADT can use many different goodness criteria to guide its search for good trees. We used three different criteria: our own max minority and sum minority and Quinlan's [1986] information gain. Averaged results are shown in Table 1.

Also shown in the table is the accuracy obtained when, for each training- and test-set pair, we take the majority vote of 11 trees when classifying the test set. Note that the accuracy, when using the majority voting scheme, is consistently higher than when using single SADT trees.

<i>Goodness Criterion</i>	<i>Average Error Rate (%)</i>	<i>Error rate with 11 trees</i>	<i>Reduction in Error</i>	<i>Best Accuracy</i>	
				<i>Error Rate</i>	<i>Number of Trees</i>
MM	5.7	4.1	28%	4.1	9
SM	5.3	3.7	30%	3.3	33
IG	5.5	4.8	13%	4.8	5

Table 1: Iris results

Weiss and Kapouleas [1989] obtained accuracies of 96.7%, 96.0%, and 95.3% with back-propagation, nearest neighbor, and CART, respectively. Their results were generated with leave-one-out trials, i.e., 150-fold cross validation.

5.1.1 Choosing a value for k

How did we choose $k = 11$ for our k -DT trees? Intuitively, it may seem that the more trees used in the voting process, the higher will be the combined accuracy. However, if an example is somehow "difficult" to classify, then voting will only make it less likely to classify that example correctly.

Figure 3 is a plot of average classification accuracy on the iris data set, as the number of trees in the voting process is varied. Note that there is a big jump in accuracy even when only three trees are used. The max minority and information gain measures peak fairly early and begin to drop off, whereas the sum minority measure is still increasing in accuracy at thirty-five trees.

We have compromised by using eleven trees, which appears to work well in practice. Table 1 shows the average classification accuracy when using eleven trees for voting. Also shown is the classification accuracy for the optimal choice of k . (The optimal choice in the table is limited by the number of cross-validation trials we have run, since we only

had that many trees to work with). The choice of 11 trees worked well for the iris dataset. The accuracy obtained with this number of trees was at least as good as any other number of trees we tried for two of the energy measures and still quite good for the third.

At this point, it is worth considering whether these results are to be expected. For each example x in the iris data set, we computed the percentage $p(x)$ of times it was correctly classified in our tests. Figure 4 shows, for a given percentage p , the fraction of the examples for which $p(x) = p$. (Note that the figure is an average over all three goodness criteria). This gives us a rough estimate on the probability of the average tree classifying that example correctly. First, note that a vast majority of the examples are always or nearly always classified correctly. Approximately, 4.4% of the examples are predicted correctly less than half of the time. These are the examples that we would expect to be classified incorrectly if we were to take a majority vote over a large number of trees. We note that this percentage is close to error rate obtained with k -DT.

5.2 Applying k -DT to cancer diagnosis

For our second experiment, we chose a dataset that has been the subject of experiments that classified the data using

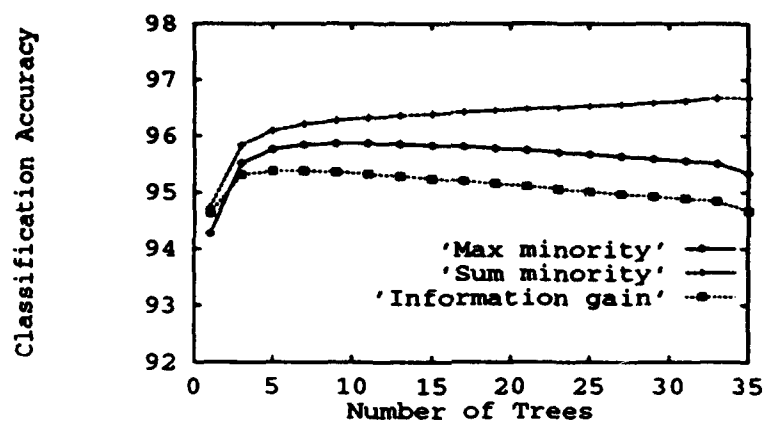


Figure 3: Iris classification accuracy vs. number of trees

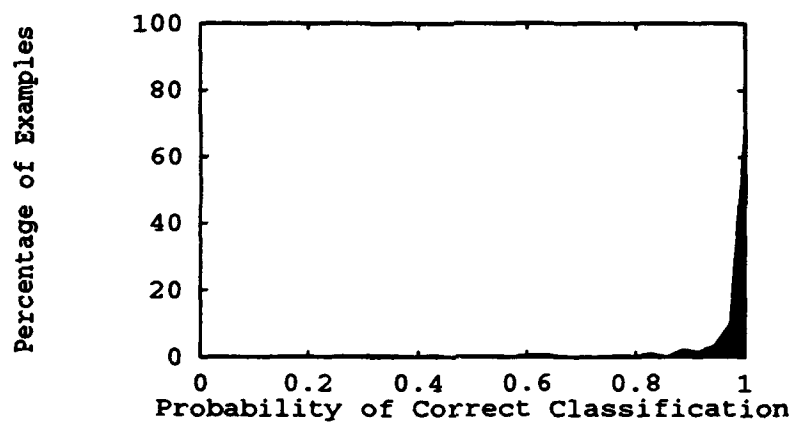


Figure 4: Percentage of iris examples achieving given accuracy

oblique hyperplanes (Mangasarian et al., 1990). This dataset contains 470 examples of patients with breast cancer, and the diagnostic task is to determine whether the cancer is benign or malignant. The input data comprised 9 numeric attributes, hence our decision trees used oblique hyperplanes in 9-D.

Mangasarian's method uses linear programming to find pairs of hyperplanes that partition the data. The algorithm finds one pair of parallel hyperplanes at a time, and each pair can be oriented at any angle with respect to all other pairs. The resulting model is a set of oblique hyperplanes, similar though not identical to an oblique decision tree.

Because Mangasarian et al. received the data as they were collected in a clinical setting, their experimental design was very simple. They trained their algorithm on the initial set of 369 examples. Of the 369 patients, 201 (54.5%) had no malignancy and the remainder had confirmed malignancies. On the next 70 patients to enter the clinic, they used their algorithm for diagnosis, and found that it correctly diagnosed 68 patients. We used $68/70 = 0.97$ as a rough estimate of the accuracy of Mangasarian et al.'s method. They then re-trained their algorithm using the 70 new patients, and reported that it correctly classified all of the next 31 patients to enter the clinic. Mangasarian reported that his program's output was being used in an actual clinical setting. Using the same dataset with a more uniform experimental design, Salzberg reported that the EACH hyper-rectangle program produced 95% classification accuracy, and 1-nearest-neighbor had 94% accuracy (Salzberg, 1991).

The results of our tests on this data are shown in Table 2. The average values are the

average of thirty-six 10-fold cross-validation trials. Once again, the accuracy obtained by using an 11-tree majority classifier is consistently higher than that of the average tree. In this example, the sum minority goodness criterion did quite a bit better on average than the other two, but it benefited less from the use of the majority technique. It is possible that by taking the majority, we are able to overcome weaknesses in the other two criteria that are not as significant with sum minority.

We also see that using eleven trees is a good choice for this dataset as well. Only for the max minority energy measure was there a noticeable difference in accuracy between the optimal choice for the number of trees and our choice of 11.

5.3 Identifying stars and galaxies

In order to study the performance of k -DT on larger datasets, we ran several experiments using astronomical image data collected with the University of Minnesota Plate Scanner. This dataset contains several thousand astronomical objects, all of which are classified as either stars or galaxies. Odewahn et al. [1992] used this dataset to train perceptrons and backpropagation networks to differentiate between stars and galaxies.

We did not have access to the exact training and test set partitions used by Odewahn et al., so we used a cross-validation technique to estimate classification accuracy. The Odewahn et al. study used a single training/test set partition. Although our results may not be completely comparable to theirs, we include them to show that both learning methods produce similar accuracies. Our results were generated by averaging nineteen 10-fold cross-validation trials.

<i>Goodness Criterion</i>	<i>Average Error Rate (%)</i>	<i>Error rate with 11 trees</i>	<i>Reduction in Error</i>	<i>Best Accuracy</i>	
				<i>Error Rate</i>	<i>Number of Trees</i>
MM	7.3	4.8	34%	4.4	33
SM	5.1	4.4	13%	4.3	23
IG	6.7	4.9	27%	4.9	11

Table 2: Breast cancer malignancy results

<i>Goodness Criterion</i>	<i>Average Error Rate (%)</i>	<i>Error rate with 11 trees</i>	<i>Reduction in Error</i>	<i>Best Accuracy</i>	
				<i>Error Rate</i>	<i>Number of Trees</i>
MM	1.2	0.5	58%	0.5	11
SM	1.0	0.7	30%	0.7	7
IG	1.0	0.6	40%	0.6	5

Table 3: Star/galaxy results

The astronomy dataset consists of 4164 examples. Each example has fourteen real-valued attributes and a label of either "star" or "galaxy." Approximately 35% of the examples are galaxies.

Classification results are shown in Table 3. Odewahn et al. [1992] obtained accuracies of 99.7% using backpropagation and 99.4% with a perceptron. It appears, however, that their results were generated with a single trial on a single partition into test- and training-set. In fact, we obtained a ten-fold cross-validated accuracy of 99.1 using a perceptron.

Using a majority classifier increased classification accuracy for this dataset, as in the other studies. For the max minority goodness criterion, we were able to reduce the error rate by almost 60%. Using eleven trees for the majority classification was a good choice for this dataset. The results for eleven trees were at least as good as for any other number of trees (up to 15, the number of cross-validation trials we ran).

5.4 Comparison with other methods

In [Heath, 1992], we explored several techniques of taking advantage of randomization in learning algorithms. Our focus in that work was on techniques that generate many trees, and use some additional criteria to select the best tree, which we then measure on the testing set. In this section, we compare those techniques to the majority classification technique.

One of our criteria for choosing the "best" tree was to choose the smallest trees. The intuition behind this technique is that smaller trees may be more concise descriptions of the problem domain, less sensitive to noise in the training data, and have a lower chance of being generated through overtraining. For each of the ten pairs of training and testing sets in a 10-fold cross-validation, we generated several SADTtrees, and then chose the smallest. We then averaged the accuracy and size of the ten chosen trees. If, for given training and testing sets, there was more than one smallest tree, we averaged them, before averaging them with the other nine.

In another experiment in [Heath, 1992] we split the training set 70/30 and trained only using 70% of the training set. The other 30% was used as a second test set. We used it to test the tree and assign it a figure of merit. We ran this several times, choosing different 70/30 splits each time and choosing the trees with the highest figures of merit. We then tested those trees on the real test set.

In Table 4, we compare k -DT with these two approaches. All three techniques gave some improvement in accuracy, although the method of choosing trees by size was not very consistent. In some cases, small trees were actually *worse* than average trees. k -DTs always performed better than using a separate test set to judge trees. It nearly always performed better than picking the smallest trees. The only exception to this was for two goodness criteria used on the iris data set. The disadvantage to k -DTs, of course, is that they are not trees, but rather collections of trees. Thus the representation created by k -DT is not as compact as with a single tree.

6 Summary of Results

We have explored the idea of using the randomization inherent in some learning techniques to advantage, by generating a number of classifiers and combining them with a majority voting scheme. We have experimented with this technique on SADT, a randomized oblique decision tree algorithm.

Our results show that majority classifiers for SADT consistently perform better than average SADT trees, which in turn perform better than standard axis-parallel trees (see Heath, 1992). The consistency and degree of improvement is better than other techniques

we have considered for increasing accuracy through randomization.

This work is preliminary; we have not tried to apply the majority technique to other types of randomized learning algorithms. However, this is a clear opportunity for future experiments. We would also like to explore combining this technique with other techniques. For example, we would like to try the majority technique on trees which are smaller than average, to see if we can get any further improvements in accuracy. It is possible that for some applications, the added complexity of a majority classifier can be a disadvantage. We are exploring ways that might allow us to combine several trees in a majority-like way, yet still end up with a small tree structure.

Acknowledgements

The authors wish to thank David Aha for providing comments and relevant references. This research was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-89-0151, and by the National Science Foundation under Grant IRI-9116843.

References

- [Breiman *et al.*, 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
- [Heath, 1992] D. Heath. *A geometric framework for machine learning*. PhD thesis, The Johns Hopkins University, Baltimore, MD, 1992.
- [Jacobs *et al.*, 1991] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mix-

Dataset	Goodness Criterion	Error Rate			
		Average	k-DT	Smallest Trees	2nd Test Set
Iris	MM	5.7	4.1	4.4	4.7
	SM	5.3	3.7	3.4	4.8
	IG	5.5	4.8	2.4	5.5
Cancer Malignancy	MM	7.3	4.8	7.8	6.5
	SM	5.1	4.4	4.9	5.2
	IG	6.7	4.9	6.8	6.8
Star/Galaxy	MM	1.2	0.5	0.8	0.8
	SM	1.0	0.7	0.9	0.5
	IG	1.0	0.6	0.9	0.7

Table 4: Majority classifier and other methods

tures of local experts. *Neural Computation*, 3:79–87, 1991.

[Lincoln and Skrzypek, 1990] W. Lincoln and J. Skrzypek. Synergy of clustering multiple back propagation networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 650–657. Morgan Kaufmann, 1990.

[Mangasarian *et al.*, 1990] O. Mangasarian, R. Setiono, and W. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. SIAM Workshop on Optimization, 1990.

[Odewahn *et al.*, 1992] S. C. Odewahn, E. B. Stockwell, R. L. Pennington, R. M. Humphreys, and W. A. Zumach. Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, 103(1):318–331, 1992.

[Quinlan, 1986] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1987] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[Salzberg, 1991] S. Salzberg. Distance metrics for instance-based learning. In

Methodologies for Intelligent Systems: 6th International Symposium, ISMIS '91, pages 399–408, New York, 1991. Springer-Verlag.

[Utgoff, 1989] P. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

[Wiess and Kapouleas, 1989] S. Wiess and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh IJCAI*, Detroit, MI, 1989. Morgan Kaufmann.

[Wolpert, 1992] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[Zhang *et al.*, 1992] X. Zhang, J. Mesirov, and D. Waltz. Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225:1049–1063, 1992.

Meta-Learning for Multistrategy and Parallel Learning

Philip K. Chan and Salvatore J. Stolfo

Department of Computer Science

Columbia University

New York, NY 10027

pkc@cs.columbia.edu and sal@cs.columbia.edu

Abstract

Meta-learning is proposed as a general technique to combine a number of separate classifiers for machine learning tasks. A number of possible approaches for meta-learning are proposed including the use of a single learning algorithm as well as a number of different learning algorithms. This paper describes meta-learning strategies for combining independently learned classifiers by different algorithms to improve overall accuracy. We also present several meta-learning strategies for combining learned classifiers by a number of parallel learning processes on subsets of training data using different algorithms. The strategies are independent of the learning algorithms used.

Key words: concept learning, meta-learning, and parallel and distributed processing.

1 Introduction

Most research in *concept learning* (or *learning from examples*) focuses on the conception and evaluation of distinct learning strategies embodied by an individual algorithm. Since different

algorithms have different representations and search heuristics, different search spaces are being explored and hence potentially diverse results can be obtained from different algorithms. Mitchell (1980) refers to this phenomenon as *inductive bias*. That is, the outcome of running an algorithm is biased in a certain direction. Furthermore, different data sets have different characteristics and the performance of different algorithms on these data sets might differ. In other words, to date there is no single algorithm that works best on all kinds of data sets. Hence, it is beneficial to build a framework that allows different learning algorithms to be used in diverse situations.

Recently, many researchers have proposed implementing learning systems by integrating in some fashion a number of different strategies and algorithms to boost overall accuracy. The basic notion behind this integration is to complement the different underlying learning strategies embodied by different learning algorithms by effectively reducing the space of incorrect classifications of a learned concept. There are many ways of integrating different learning algorithms. For example, work on integrating concept and explanation-based learning (Flann

& Dietterich, 1989; Towell et al., 1990) requires a complicated new algorithm that implements both approaches to learning in a single system. Another line of work focuses on combining different learning systems in a loose fashion. For example, Silver et al.'s (1990) work on using a coordinator to gather votes from three different learners and Holder's (1991) work on selecting learning strategies based on their relative utility. One advantage of this approach is its simplicity in treating the individual learning systems as black boxes with little or no modification required to achieve a final system. Therefore, individual systems can be added or replaced with relative ease.

A more interesting approach to loosely combine learners is to learn how to combine independently learned concepts. Stolfo et al.'s work (1989) attempts to learn rules for merging different phoneme output representations from multiple trained speech recognizers. Wolpert (1992) presents a theory on *stacked generalization (meta-learning)*. Several (level 0) classifiers are first learned from the same training set. The predictions made by these classifiers on the training set and the correct classifications form the training set of the next level (level 1) classifier. When an instance is being classified, the level 0 classifiers first make their predictions on the instance. The predictions are then presented to the level 1 classifier, which makes the final prediction. Zhang et al.'s (1992) work utilizes a similar approach to learn a *combiner* based on the predictions made by three different classifiers.

Furthermore, much of the research in concept learning concentrates on problems with relatively small amounts of data. With the coming age of "high-capacity" and "light-speed" networks, it is likely that orders of magnitude more data will be available for various learning problems of real world importance. The Grand Challenges of HPCC (Wah et al., 1993)

are perhaps the best examples.

Quinlan (1979) approached the problem of efficiently applying learning systems to data that are substantially larger than available memory with a windowing technique. A learning algorithm is applied to a small subset of training data, called a *window*, and the learned concept is tested on the remaining training data. This is repeated on a new window of the same size with some of the incorrectly classified data replacing some of the data in the old window until all the data are correctly classified. Wirth and Catlett (1988) show that the windowing technique does not significantly improve speed on reliable data. On the contrary, for noisy data, windowing considerably slows down the computation. Catlett (1991) demonstrates that larger amounts of data improves accuracy, but he projects that ID3 (Quinlan, 1986) on modern machines will take several months to learn from a million records in the flight data set obtained from NASA. He proposes some improvements to the ID3 algorithm, but his scheme is limited to real-numbered attributes and the complexity is still prohibitive for large amounts of data (Chan & Stolfo, 1993c). Typical learning systems like ID3 cannot handle data that exceed the size of a monolithic memory on a single processor. We believe parallel and distributed processing with *divide-and-conquer* techniques provides the best hope of dealing with such large amounts of data in terms of speed and memory requirement. But how precisely does one organize and implement a parallel processing system for machine learning tasks?

One approach to this problem is to parallelize the learning algorithms. Zhang et al.'s (1989) work on parallelizing the backpropagation algorithm on a Connection Machine is one example. This approach requires optimizing the code of a particular algorithm for a specific architecture. Our approach is to run the serial code on a number of data subsets in parallel and combine the

results in an intelligent fashion. This approach has the advantage of using the same serial code without the time-consuming process of parallelizing it. In addition, our proposed framework for combining the results of learned concepts is independent of the learning algorithms and the computing platform used. However, this approach cannot guarantee the accuracy of the learned concepts to be the same as the serial version since a considerable amount of information may not be accessible to each of the learners.

In this paper we present the concept of *meta-learning*, introduced in (Chan & Stolfo, 1993c), and its use in coalescing the results from multiple concept learning systems to improve accuracy and the results achieved from a set of parallel or distributed learning processes to improve learning speed. The ultimate goal of this work is to improve both the accuracy and efficiency of machine learning by means of parallel processing of multiple learning systems applied to massive amounts of training data. There are many ways one might imagine to combine learned classifiers. For this paper, we detail only a few approaches. Thus, this work may be viewed as exploratory to determine the efficacy of the general approach. Section 2 discusses meta-learning and how it facilitates multistrategy and parallel learning. Section 3 details our strategies for boosting accuracy by meta-learning, which appear in (Chan & Stolfo, 1993a). Section 4 discusses our approach to improve speed and accuracy through meta-learning. Section 5 concludes with our findings and work in progress.

2 Meta-learning

Meta-learning can be loosely defined as learning from information generated by a learner(s). It can also be viewed as the learning of meta-knowledge on the learned information. In our work we concentrate on learning from the out-

put of concept learning systems. In this case meta-learning means learning from the *classifiers* produced by the learners and the *predictions* of these classifiers on *training data*. A classifier (or concept) is the output of a concept learning system and a prediction (or classification) is the predicted class generated by a classifier when an instance is supplied. That is, we are interested in the output of the learners, not the learners themselves. Moreover, the training data presented to the learners initially are also available to the *meta-learner* if warranted.

In essence we use multiple strategies to improve accuracy and parallelism to improve speed. The use of meta-learning can facilitate reaching these goals. This is demonstrated by four frameworks supported by meta-learning as follows:

1. *Hypothesis boosting (HB)* involves the improvement of accuracy of a learning algorithm by meta-learning. A number of instances of a single learning algorithm are applied to distinguished subsets of training data that are composed in such a way as to improve the overall prediction accuracy (Schapire (1990) calls this hypothesis boosting). Based on an initial learned hypothesis for some concept derived from a random distribution of training data, Schapire (1990) generates two additional distributions of examples, to which the learning algorithm is then applied. The three different distributions are interrelated and generated successively. The predictions of the three learned classifiers are combined using a simple arbitration rule. Although his approach is limited to the PAC model of learning, some success was achieved in the domain of character recognition, using neural networks (Drucker et al. 1993). Freund (1990) has a similar approach, but with potentially many more distributions. This framework focuses pri-

marily on improving the accuracy of an individual learner.

2. *Parallel learning (PL)* involves applying a single algorithm on different subsets of the data in parallel and the use of meta-learning to combine the partial results. Unlike *hypothesis boosting*, the subsets are independent and can be generated concurrently. This approach attempts to improve speed, not accuracy, via parallelism. Not much work by others has been done in applying meta-learning to parallel learning.
3. *Multistrategy hypothesis boosting (MSHB)* involves applying multiple algorithms on the same set of data and the results of the learned concepts are combined by meta-learning to improve overall accuracy. The aforementioned approaches used by Wolpert (1992) and Zhang et al. (1992) are examples of this strategy. This approach attempts to take advantage of the diversity of learners to increase accuracy.
4. *Multistrategy parallel learning (MSPL)* is a combination of *parallel learning* and *multistrategy hypothesis boosting*. Multiple learning algorithms are applied to subsets of the data in parallel. This framework tries to improve both speed via parallelism and accuracy via diversity. To our knowledge, not much research by others has been attempted in this framework, other than the proposed work in (Stolfo et al., 1989).

Our work has concentrated on *parallel learning* and *multistrategy hypothesis boosting*, which, we believe, will provide some insights in how to achieve *multistrategy parallel learning*. In the rest of this paper we will discuss our approach to the multistrategy hypothesis boosting and multistrategy parallel learning frameworks.

3 Multistrategy Hypothesis Boosting

The objective here is to improve prediction accuracy by exploring the diversity of multiple learning algorithms through meta-learning. This is achieved by a basic configuration which has several different *base learners* and one *meta-learner* that learns from the output of the base learners. The meta-learner may employ the same algorithm as one of the base learners or a completely distinct algorithm. Each of the base learners is provided with the entire training set of raw data. However, the training set for the meta-learner varies according to the strategies described below, and is quite different than the data used in training the base classifiers. Note that the meta-learner does not aim at picking the "best" classifier generated by the base learners; instead it tries to combine the classifiers. That is, the prediction accuracy of the overall system is not limited to the most accurate base learner. It is our intent to generate an overall system that outperforms the underlying base learners.

There are in general two types of information the meta-learner can combine: the learned *base classifiers* and the *predictions* of the learned base classifiers. The first type of information consists of *concept descriptions* in the base classifiers (or concepts). Some common concept descriptions are represented in the form of decision trees, rules, and networks. Since we are aiming at diversity in the base learners, the learning algorithms chosen usually have different representations for their learned classifiers. Hence, in order to combine the classifiers, we need to define a common representation to which the different learned classifiers are translated. However, it is difficult to define such a representation to encapsulate all the representations without losing a significant amount of information during the translation process. For instance, it is very difficult to define a com-

mon representation to integrate the discriminant functions and exemplars computed by a nearest-neighbor learning algorithm with the tree computed by a decision tree learning algorithm. Because of this difficulty, one might define a uniform representation that limits the types of representation that can be supported and hence the choice of learning algorithms.

An alternative strategy is to integrate the *predictions* of the learned classifiers for the training set leaving the internal organization of each classifier completely transparent. These predictions are hypothesized classes present in the training data and can be categorical or associated with some numeric measure (e.g., probabilities, confidence values, and distances). In this case, the problem of finding a common ground is much less severe. For instance, classes with numeric measures can be treated as categorical (by picking the class with the highest value). Since any learner can be employed in this case, we focus our work on combining predictions from the learned classifiers. Moreover, since converting categorical predictions to ones with numeric measures is undesirable or impossible, we concentrate on combining categorical predictions.

We experimented with three types of meta-learning strategies (*combiner*, *arbiter*, and *hybrid*) for combining predictions, which we discuss in the following sections. For pedagogical reasons, our discussion assumes three base learners and one meta-learner.

3.1 Combiner strategy

In the *combiner* strategy, the predictions for the training set generated by a two-fold cross validation technique using the base learners form the basis of the meta-learner's training set (details in (Chan & Stolfo, 1993a)). A *composition rule*, which varies in different schemes, determines the content of training examples for the meta-learner. From these examples, the meta-

learner generates a meta-classifier, that we call a *combiner*. In classifying an instance, the base classifiers first generate their predictions. Based on the same composition rule, a new instance is generated from the predictions, which is then classified by the combiner. The aim of this strategy is to coalesce the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction.

We experimented with three schemes for the composition rule. First, three predictions, $C_1(x)$, $C_2(x)$, and $C_3(x)$, for each example x in the original training set of examples, E , are generated by three separate classifiers, C_1 , C_2 , and C_3 . These predicted classifications are used to form a new set of "meta-level training instances," T , which is used as input to a learning algorithm that computes a combiner. The manner in which T is computed varies according to the schemes as defined below. In the following definitions, $class(x)$ denotes the correct classification of example x as specified in training set E .

1. Return meta-level training instances with the correct classification and the predictions; i.e., $T = \{(class(x), C_1(x), C_2(x), C_3(x)) \mid x \in E\}$. This scheme was also used by Wolpert (1992). (For further reference, this scheme is denoted as *meta-class*.) A sample training set is depicted in Figure 1.
2. Return meta-level training instances similar to those in the first (*meta-class*) scheme with the addition of the original attribute vectors in the training examples; i.e., $T = \{(class(x), C_1(x), C_2(x), C_3(x), attrvec(x)) \mid x \in E\}$. (Henceforth, this scheme is denoted as *meta-class-attribute*.)
3. Return meta-level training instances similar to those in the *meta-class* scheme except that each prediction, $C_i(x)$, has

m binary predictions, $C_{i_1}(x), \dots, C_{i_m}(x)$, where m is the number of classes. Each prediction, $C_{i_j}(x)$, is produced from a binary classifier, which is trained on examples that are labeled with classes j and $\neg j$. In other words, we are using more specialized base classifiers and attempting to learn the correlation between the binary predictions and the correct prediction. For concreteness, $T = \{(class(x), C_{1_1}(x), \dots, C_{1_m}(x), C_{2_1}(x), \dots, C_{2_m}(x), C_{3_1}(x), \dots, C_{3_m}(x)) \mid x \in E\}$. (Henceforth, this scheme is denoted as *meta-class-binary*.)

These three schemes for the composition rule are defined in the context of forming a training set for the combiner. These composition rules are also used in a similar manner during classification after a combiner has been computed. Given a test instance whose classification is sought, we first compute the classifications predicted by each of the base classifiers. The composition rule is then applied to generate a single meta-level test instance, which is then classified by the combiner to produce the final predicted class of the original test instance.

3.2 Arbiter strategy

In the *arbiter* strategy, the training set for the meta-learner is a subset of the training set for the base learners. That is, the meta-level training instances are a particular distribution of the raw training set E . The predictions of the learned base classifiers for the training set and a *selection rule*, which varies in different schemes, determines which subset will constitute the meta-learner's training set. (This contrasts with the *combiner* strategy which has the same number of examples for the base classifier as for the combiner. Also, the meta-level instances of the combiner strategy incorporate additional information than just the raw training data.) Based on this training set, the meta-learner generates a

meta-classifier, in this case called an *arbiter*. In classifying an instance, the base classifiers first generate their predictions. These predictions, together with the arbiter and a corresponding *arbitration rule*, generate the final predictions. (This contrasts with the multi-level arbiter trees discussed in Section 4.1.) In this strategy one learns to arbitrate among the potentially different predictions from the base classifiers, instead of learning to coalesce the predictions as in the combiner strategy. We first describe the schemes for the selection rule and then those for the arbitration rule.

We experimented with two schemes for the *selection rule*, which chooses training examples for an arbiter. In essence the schemes select examples that are confusing to the three base classifiers, from which an arbiter is learned. Based on three predictions, $C_1(x)$, $C_2(x)$, and $C_3(x)$, for each example x in a set of training examples, E , each scheme generates a set of training examples, $T (\subseteq E)$, for the arbiter. The two versions of this selection rule implemented and reported here include:

1. Return instances with predictions that disagree; i.e., $T = T_d = \{x \in E \mid (C_1(x) \neq C_2(x)) \vee (C_2(x) \neq C_3(x))\}$. Thus, instances with conflicting predictions are used to train the arbiter. However, instances with predictions that agree but are incorrect are not included. (We refer to this scheme as *meta-different*.) A sample training set is depicted in Figure 1.
2. Return instances with predictions that disagree, T_d , as in the first case (*meta-different*), but also instances with predictions that agree but are incorrect; i.e., $T = T_d \cup T_i$, where $T_i = \{x \in E \mid (C_1(x) = C_2(x) = C_3(x)) \wedge (class(x) \neq C_1(x))\}$. Note that we compose both cases of instances that are incorrectly classified or are in disagreement. (Henceforth, we refer to this case as *meta-different-incorrect*.)

Class	Attribute vector	Example	Base classifiers' predictions		
$class(x)$	$attrvec(x)$	x	$C_1(x)$	$C_2(x)$	$C_3(x)$
a	$attrvec_1$	x_1	a	a	a
b	$attrvec_2$	x_2	a	b	c
c	$attrvec_3$	x_3	c	b	a

Training set from the <i>meta-class</i> combiner scheme		
Instance	Class	Attribute vector
1	a	(a, a, a)
2	b	(a, b, c)
3	c	(c, b, a)

Training set from the <i>meta-different</i> arbiter scheme		
Instance	Class	Attribute vector
1	b	$attrvec_2$
2	c	$attrvec_3$

Figure 1: Sample training sets generated by the *combiner* and *arbiter* strategies

The arbiters are trained by some learning algorithm on the particular distinguished distributions of training data and are used in generating predictions. During the classification of an instance, y , the learned arbiter, A , and the corresponding *arbitration rule*, produce a final prediction based on the three predictions, $C_1(y)$, $C_2(y)$, and $C_3(y)$, from the three base classifiers and the arbiter's own prediction, $A(y)$. The following arbitration rule applies to both schemes for the selection rule described above.

- 1&2. Return the simple vote of the base and arbiter's predictions, breaking ties in favor of the arbiter's prediction; i.e., if there are no ties, return $vote(C_1(y), C_2(y), C_3(y), A(y))$, otherwise return $A(y)$.

3.3 Hybrid strategy

We integrate the *combiner* and *arbiter* strategies in the *hybrid* strategy. Given the predictions of the base classifiers on the original training set, a *selection rule* picks examples from the training set as in the arbiter strategy. However, the training set for the meta-learner is generated by a *composition rule* applied to the distribution of training data (a subset of E) as defined in the combiner strategy. Thus, the hybrid strategy attempts to improve the arbiter strategy by

correcting the predictions of the "confused" examples. It does so by using the combiner strategy to coalesce the predicted classifications of instances in disagreement by the base classifiers, instead of purely arbitrating among them. A learning algorithm then generates a meta-classifier from this training set. When a test instance is classified, the base classifiers first generate their predictions. These predictions are then composed to form a meta-level instance for the learned meta-classifier using the same composition rule. The meta-classifier then produces the final prediction.

We experimented with two combinations of composition and selection rules, though any combination of the rules is possible:

1. Select examples that have different predictions from the base classifiers and the predictions, together with the correct classes and attribute vectors from the training set for the meta-learner. This integrates the *meta-different* and *meta-class-attribute* schemes. (Henceforth, we refer to this scheme as *meta-different-class-attribute*.)
2. Select examples that have different or incorrect predictions from the base classifiers and the predictions, together with the correct classes and attribute vectors form

the training set for the meta-learner. This integrates the *meta-different-incorrect* and *meta-class-attribute* schemes. (Henceforth, denoted as *meta-different-incorrect-class-attribute*.)

We discussed three general meta-learning strategies (*combiner*, *arbiter*, and *hybrid*) for *multistrategy hypothesis boosting*. The combiner strategy aims at coalescing predictions from the constituent classifiers, whereas the arbiter strategy arbitrates among them. In addition, the training set for the combiner strategy includes examples derived from the entire original training set, whereas the one for the arbiter includes only examples chosen by a selection rule from the original set. That is, the training set for the arbiter strategy is usually smaller than the one for the combiner strategy and hence contains less information. The hybrid strategy is intended to augment this deficiency in the arbiter strategy by coalescing the predictions from the selected examples. We postulate that the combiner strategy would still be the most effective one due to the larger amount of information available and the coalescing process.

Among the combiner schemes, the *meta-class-attribute* scheme provides more information (the addition of attribute vectors) than the *meta-class* scheme in the combiner training set. The *meta-class-binary* scheme provides more precise information for the meta-learner because more specialized base classifiers are used. Among the arbiter schemes, the *meta-different-incorrect* scheme includes more examples in the arbiter training set than the *meta-different* scheme. Similar observations can be made for corresponding schemes in the hybrid strategy.

The choice of the meta-learner to perform the above strategies is another issue. Due to the relatively low regularity in the training data for meta-learners, we postulate that a probabilistic learner would be more effective than a categor-

ical one.

3.4 Summary of empirical results

Experiments on the aforementioned strategies were run with different combinations of three base-learners and one meta-learner. In the experiments we employed four different learning algorithms: BAYES (described in (Clark & Niblett, 1987)), ID3 (Quinlan, 1986), CART (Breiman et al., 1984, and WPEBLS (the weighted version of PEBLS (Cost & Salzberg, 1993)), and two molecular biology data sets: protein secondary structures (SS) (Qian & Sejnowski, 1988) and DNA splice junctions (SJ) (Towell et al., 1990). Details of the experiments and quantitative results obtained are reported in (Chan & Stolfo, 1993a). Space limitations prevent us from displaying them here. We summarize our results as follows.

There are two ways to analyze the results. First, we consider whether the employment of a meta-learner improves accuracy with respect to the underlying three base classifiers. For both sets of data, we discovered improvements were always achieved when BAYES was used as the meta-learner and the other three learning algorithms (ID3, CART, and WPEBLS) served as the base-learners (improved from 55.4% up to 60.7% in SS and from 94.8% up to 96.6% in SJ), regardless of the meta-learning strategies employed. Next, when we considered combinations of a particular meta-learner and strategy, regardless of the base learners, the results were mixed. For the SJ data set, the same or better accuracy was consistently attained when BAYES was the meta-learner in the *meta-class-attribute* strategy (improved from 94.8% up to 97.2%) and ID3 in the *meta-class* and *meta-class-attribute* strategies (improved from 94.8% up to 96.9%), regardless of the base learners. For the SS data set, none of the meta-learner/strategy combinations maintained a consistent improvement.

Second, we consider whether the use of meta-learning achieves higher accuracy than the most accurate single-strategy learner (which was BAYES in this case). For the SJ data set, improvement was consistently achieved when BAYES is the meta-learner in the *meta-class-attribute* strategy (improved from 96.4% up to 97.6%), regardless of the base learners. In fact, when the base learners are BAYES, ID3, and CART, the overall accuracy was the highest obtained. For the SS data set, almost all the results did not outperform BAYES as a single-strategy learner.

The two data sets chosen represent two different kinds of data sets: SS is difficult to learn (50+% accuracy) and SJ is easy to learn (90+% accuracy). Our experiments indicate that some of our meta-learning strategies improve accuracy in the SJ data set. However, in the SS data set, meta-learning did not improve accuracy. This can be attributed to the quality of predictions from the base classifiers for the two data sets. The high percentage of having one or none correct out of three predictions in the SS data set might greatly hinder the ability of meta-learning to work. One possible solution is to increase the number of base classifiers to lower the percentage of having one or none correct predictions.

In general, the *combiner* strategies performed more effectively than the *arbiter* and *hybrid* strategies in the test cases studied. To our surprise, the hybrid schemes did not improve the arbiter strategies. This indicates that coalescing the predictions are more beneficial than arbitrating among them. Among the combiner strategies, *meta-class-attribute* was particularly effectively. This suggests that predictions alone are not sufficient for meta-learning. Furthermore, BAYES was usually the more successful meta-learner, which coincides with our earlier intuition that probabilistic meta-learners might be more effective than others.

4 Multistrategy Parallel Learning

The dual objectives of the *multistrategy parallel learning* (MSPL) framework are to improve accuracy using multiple algorithms and to speed up the learning process by parallel processing in a *divide-and-conquer* fashion. Since the MSPL framework is an integration of the *multistrategy hypothesis boosting* (presented in the previous section) and the *parallel learning* (PL) frameworks, we briefly review the PL framework introduced in (Chan & Stolfo, 1993c) before we discuss the MSPL framework.

4.1 Parallel learning

The objective here is to speed up the learning process by *divide-and-conquer*. The data set is partitioned into subsets and the same learning algorithm is applied on each of these subsets. Several issues arise here.

First, how many subsets should be generated? This largely depends on the number of processors available and the size of the training set. The number of processors puts an upper bound on the number of subsets. Another consideration is the desired accuracy we wish to achieve; there may be a tradeoff between the number of subsets and the final accuracy. Moreover, the size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective classifier.

Second, what is the distribution of training examples in the subsets? The subsets can be disjoint or overlap. The class distribution can be random, or follow some deterministic scheme. We experimented with disjoint equal-size subsets with random distributions of classes.

Third, how do we apply meta-learning to coalescing partial results generated by the learning processes? This is the more important ques-

tion. Our approach is meta-learning *arbiters* in a binary-tree fashion.

Based upon a number of candidate predictions, an arbiter, together with an *arbitration rule*, decides a final outcome (similar to the arbiter strategy described in Section 3.2). An arbiter is learned from the output of a pair of learning processes and recursively, an arbiter is learned from the output of two arbiters. A binary tree of arbiters (called an *arbiter tree*) is generated with the initially learned classifiers at the leaves. For s subsets and s classifiers, there are $\log_2(s)$ levels in the generated arbiter tree. The manner in which an arbiter tree is computed and used is the subject of the following sections.

4.2 Classifying using an arbiter tree

When an instance is classified by the arbiter tree, predictions flow from the leaves to the root. First, each of the leaf classifiers produces an initial prediction; i.e., a classification of the test instance. From a pair of predictions and the parent arbiter's prediction, a combined prediction is produced by an *arbitration rule*. This process is applied at each level until a final prediction is produced at the root of the tree. Detailed schemes for the arbitration rule are reported in (Chan & Stolfo, 1993c) and similar ones can be found in Section 3.2. Since at each level, the leaf classifiers and arbiters are independent, predictions are generated in parallel. Further issues and strategies for efficiently generating predictions by arbiter trees are beyond the scope of this paper. Next, we describe how an arbiter tree is learned.

4.3 Meta-learning an arbiter tree

We experimented with several schemes to meta-learn a binary tree of arbiters. In all these schemes the leaf classifiers are first learned from randomly chosen disjoint data subsets and

the classifiers are grouped in pairs. (The strategy for pairing classifiers is the subject of future study and is discussed later.) For each pair of classifiers, the union of the data subsets on which the classifiers are trained is generated. This union set is then classified by the two classifiers. A *selection rule* compares the predictions from the two classifiers and selects instances from the union set, which form the training set for the arbiter of the pair of classifiers. Thus, the rule acts as a data filter to produce a training set with a particular distribution. Detailed strategies for the selection rule are reported in (Chan & Stolfo, 1993c) and similar schemes can be found in Section 3.2. The arbiter is learned from this set with the same learning algorithm. The process of forming the union of data subsets, classifying it using a pair of arbiter trees, comparing the predictions, forming a training set, and training the arbiter is recursively performed until the root arbiter is formed.

For example, suppose there are initially four training data subsets ($T_1 - T_4$). First, four classifiers ($C_1 - C_4$) are generated in parallel from $T_1 - T_4$. The union of subsets T_1 and T_2 , U_{12} , is then classified by C_1 and C_2 , which generates two sets of predictions (P_1 and P_2). Based on predictions P_1 and P_2 , and the subset U_{12} , a selection rule generates a training set (T_{12}) for the arbiter. The arbiter (A_{12}) is then trained from the set T_{12} using the same learning algorithm used to learn the initial classifiers. Similarly, arbiter A_{34} is generated in the same fashion starting from T_3 and T_4 , in parallel with A_{12} , and hence all the first-level arbiters are produced. Then U_{14} is formed by the union of subset T_1 through T_4 and is classified by the arbiter trees rooted with A_{12} and A_{34} . Similarly, T_{14} and A_{14} (root arbiter) are generated and the arbiter tree is completed (see Figure 2).

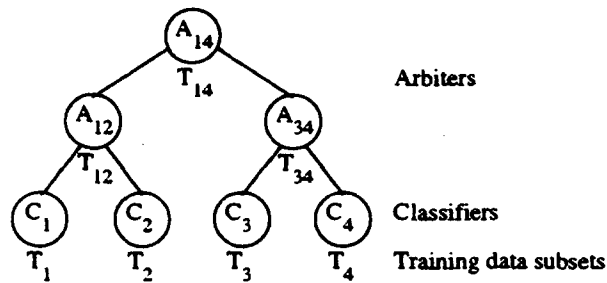


Figure 2: Sample arbiter tree

4.3.1 Summary of empirical results

Experiments on the aforementioned strategies were run on the four different learning algorithms and two data sets described in the Section 3.4. Details of the experiments and preliminary results obtained from a serial implementation are reported in (Chan & Stolfo, 1993c). Here we summarize those results.

In one set of experiments, we restricted the training set size for an arbiter to be no larger than the training set size for a leaf classifier. Hence, the amount of computation in training an arbiter is bounded by the time to train a leaf classifier. In a parallel computation model each level of arbiters can be learned as efficiently as the leaf classifiers, and hence significant speed up can be predicted. Our arbiter schemes maintained the low accuracy in the SS data set and degraded (up to 10% with 32 subsets) the high accuracy (90+%) in the SJ data set. The accuracy drop in the SJ data can be attributed the absence of a particular class in half of the arbiter tree, resulting from a random class distribution in the subsets. Another reason might be the small size of each subset, which has about 80 examples.

When the restriction on the size of the training set for an arbiter is lifted, the same level of accuracy can be achieved. For the SJ data set, empirical results show that the single largest arbiter training set is about 30% of the en-

tire training set. Moreover, up to a six-fold speed up with eight subsets/processors was still achieved based on a theoretical calculation using the $O(n^2)$ model of WPEBLS. (Not much improvement can be obtained in linear algorithms like BAYES.) That is, less time and memory than a serial version is needed to reach the same results. However, the amount of speed up leveled off after eight subsets because the largest arbiter training set appeared at the root, which formed a bottleneck. Since the pairing of classifiers/arbiters affects the arbiter training set sizes, we are currently investigating pairing strategies to reduce the size of the largest set. One scheme is to pair classifiers and arbiters that agree most often with each other. Another scheme is to pair those that disagree the most. At first glance the first scheme would seem to be more attractive. However, since disagreements are present, if they do not get resolved at the bottom of the tree, they will all surface near the root of the tree, which is also when the choice of pairings is limited or nonexistent (there are only two arbiters one level below the root). Hence, it might be more beneficial to resolve conflicts near the leaves leaving fewer disagreements near the root. Empirical results indeed show that pairing the classifiers that produced the larger sets (more disagreements) at the leaf level reduced the size of the largest set in the tree.

4.4 Approaches to MSPL

Recall, we seek to both improve accuracy and speed up the computation. There are three general approaches in achieving these goals that differ in the amount of data used and the manner in which the learning algorithms are applied. For pedagogical reasons, the following discussion will use the *combiner* approach as the default scheme for combining results at the meta-level. Here we present the approaches currently under development and some empirical results obtained for the second approach.

4.4.1 Coarse-grain diversity with the entire training set

In this approach each learning algorithm is applied to the entire training data set. The learners are run concurrently and each follows the PL framework. As a result, each learner produces an arbiter tree. The predictions of the arbiter trees on the training set become part of the training examples for the combiner in the MSHB framework. Since the training of the combiner has to be performed after the arbiter trees are formed, all the processors used to train the arbiter trees will be available for training the combiner. Again, the combiner is trained using the PL framework, which generates another arbiter tree. During the classification of an instance, predictions are generated from the learned arbiter trees and are coalesced by the learned combiner (an arbiter tree itself). For further reference, this approach is denoted as *coarse-all*. Figure 3(a) schematically depicts the arbiter trees formed. Each triangle represents a learned arbiter tree. L_1, L_2, L_3 are the different learners (three of them in this case) and L_c is the learner for the combiner. E is the original set of training examples. Since this approach is similar to the MSHB framework, except the use of arbiter trees, accuracy results similar to those presented in Section 3.4 can be expected here.

Since different learning algorithms have different complexity, different learners will finish the same task at different times. That is, it is important to determine how many processors are allocated to each learner to balance the computational load and reduce any large variance in completion times. To define and implement a scheme to allocate processors and data, relative speeds of the learners have to be measured. One approach is to determine the speed of a learner empirically relative to the data set size and derive a function to approximate the relationship between speed and data set size. Rel-

ative speeds of the learners are then estimated by these functions. Given the relative speeds, we then allocate processors to each learner to achieve load balancing.

4.4.2 Coarse-grain diversity with data subsets

This approach is similar to the previous one except that each learner is applied concurrently to a different data subset, rather than the entire data set. That is, the data set is divided into l subsets, where l is the number of learning algorithms available, and a different learning algorithm is applied to each subset. The PL framework is then applied to each algorithm-subset pair. As a result, l arbiter trees will be formed. The predictions of the arbiter trees on the training set become part of the training examples for the combiner in the MSHB framework. Similar to the previous approach, the combiner is generated as another arbiter tree. When an instance is classified, the learned arbiter trees first produce their predictions, which are then coalesced by the learned combiner. This approach is denoted as *coarse-subset*. Figure 3(b) schematically depicts the arbiter trees formed. L_1, L_2, L_3 are the different learners and L_c is the learner for the combiner. E_1, E_2, E_3 are subsets of the original set of training examples.

As in the previous approach, load balancing is essential in minimizing the overall training time. However, in this approach we have to determine how to allocate the data subsets as well as the processors. One approach is to evenly distribute the data among the learners and allocate processors according to their relative speeds. Another approach is that each learner has the same number of processors and data are distributed accordingly. That is, we have to decide whether we allocate a uniform number of processors or a uniform amount of data to each learner. Since the amount of data affects the quality of the learned concepts, it

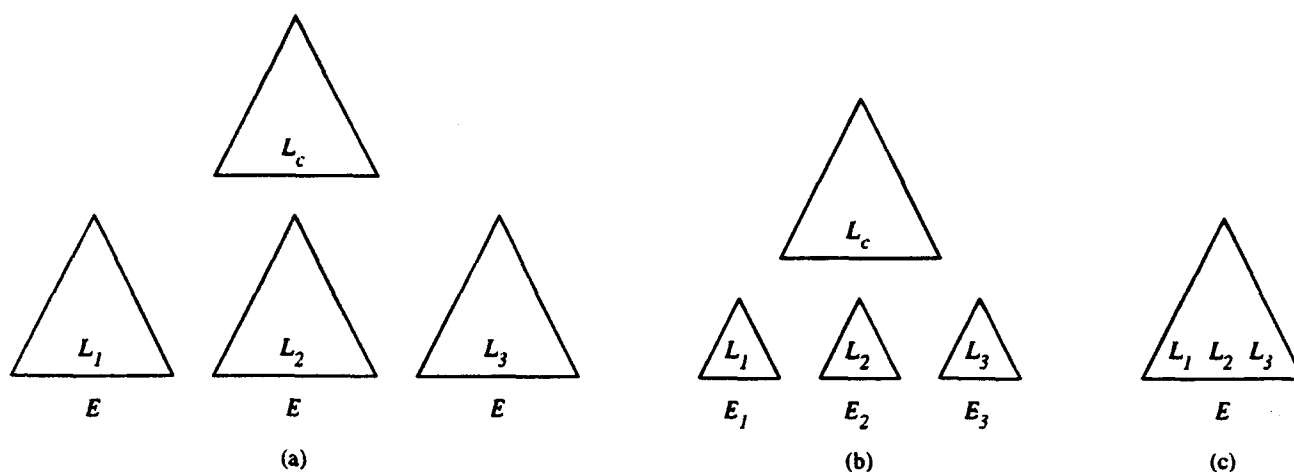


Figure 3: Learned arbiter trees from the MSPL approaches with three different learners

is more desirable to evenly distribute the data so that the learners are not biased at this stage. That is, slower learners should not be penalized with less information and thus they should be allocated more processors.

This raises the question of whether data should be distributed at all; that is, should each learner have all the data, as in the previous approach? Obviously, if each learner has the entire set of data, it would be slower than when it has only a subset of the data. It is also clear that the more data each learner has, the more accurate the generated concepts will be. That is, it is a tradeoff between speed and quality. But in problems with very large databases, we may have no choice but to distribute subsets of the data. Another question is what the data distribution is for the data subsets. The subsets can be disjoint or overlapped according to some scheme. We prefer disjoint subsets because it allows the maximum degree of parallelism. The classes represented in the subsets can be distributed randomly, uniformly, or according to some scheme. Since maintaining the same class distribution in each subset as in the entire set does not create the potential problem of missing classes in certain subsets, it is our preferred distribution scheme.

Summary of empirical results Experiments on the *coarse-subset* approach with the strategies discussed in Section 3 were run on the four different learning algorithms and two molecular biology data sets described in the Section 3.4. Details of the experiments and preliminary quantitative results obtained from a serial implementation without the use of arbiter trees are reported in (Chan & Stolfo, 1993b). We summarize those results as follows.

There are three ways to analyze the results. First, we look at whether the employment of a meta-learner improves accuracy with respect to the underlying three base classifiers learned on a subset. For the SJ data, improvements were almost always achieved when the combinations of base learners are ID3-CART-WPEBLS (improved from 94.1% up to 96.2%) and BAYES-CART-WPEBLS (from 95.7% up to 97.2%), regardless of the meta-learners and strategies. For the SS data, when the combination of base learners is ID3-CART-WPEBLS, more than half of the meta-learner/strategy combinations achieved higher accuracy than any of the base learners (from 53.9% up to 96.2%).

Second, we look at whether the use of meta-learning achieves higher accuracy than the most accurate classifier learned from a subset

(BAYES in this case). For the SJ data, the *meta-class-attribute* strategy with BAYES as the meta-learner always attained higher accuracy (from 95.7% up to 97.2%), regardless of the base learners and strategies. For the SS data, all the results did not outperform BAYES as a single base learner.

Third, we look at whether the use of meta-learning achieves higher accuracy than the most accurate classifier learned from the full training set (BAYES in this case). For the SJ data, *meta-class-attribute* strategy with BAYES as the meta-learner almost always attained higher accuracy (from 96.4% up to 97.2%), regardless of the base learners and strategies. For the SS data, all the results did not outperform BAYES.

As in the results from the MSHB framework (Section 3.4), *meta-class-attribute* is the more effective scheme and BAYES is the more successful meta-learner. Therefore, it reinforces our conjecture that coalescing results are more effective than arbitrating among them and predictions alone are not enough for meta-learning. Compared to results obtained from the MSHB framework, smaller improvements were observed here. This is mainly due to the smaller amount of information presented to the base learners.

4.4.3 Fine-grain diversity

In this approach the data set is divided into p subsets, where p is the number of processors available, and a different learning algorithm is applied to each subset. The subsets are paired and an arbiter tree is formed in a similar fashion as for the single-strategy arbiter tree. That is, instead of using the same algorithm for training an arbiter and its two children as in the PL framework, different algorithms are used. This also results in generating only one arbiter tree, which contrasts with generating multiple arbiter trees in the previous two approaches. When

an instance is classified, the prediction of the learned arbiter tree is the final prediction. This approach is denoted as *fine-grain*. Figure 3(c) schematically depicts the arbiter tree formed. L_1, L_2, L_3 are the different learners. E is the original set of training examples.

Since each subset is allocated to one processor and different algorithms have different speeds, the size of the subsets needs to be adjusted accordingly to achieve load balancing. That is, the size of the p subsets is determined by the relative speeds of different algorithms applied to the subsets. Another issue we consider is how the learning algorithms are allocated to the p subsets and the subsequent training sets for the arbiters. They could be allocated uniformly, randomly, or according to some scheme.

4.4.4 Discussion

In terms of learning speed, the *coarse-grain* approaches are slower than the *fine-grain* approach because the *combiner* has to be learned after all the base arbiter trees are learned; however, only one arbiter tree is learned in the *fine-grain* approach. In terms of overall prediction accuracy, the *coarse-all* approach should be more accurate than the *coarse-subset* approach because of the larger amount of information available to each learner in the first approach. (Recall that each arbiter tree in *coarse-all* is trained on the entire training set, which contrasts to a subset in *coarse-subset*.) It is unclear at this point how the *fine-grain* approach will perform compared to the other two and is the subject of further experimentation. Furthermore, the *coarse-grain* approaches use a *combiner* to coalesce the arbiter trees, whereas the *fine-grain* approach does not. However, more diversity is present in the *fine-grain* approach.

In our experiments for the MSHB framework, one of the learning algorithms (BAYES) consistently performs better as a meta-learner than

the other algorithms. (Recall, meta-learners learn from the output of other learners and base learners learn from the initial/raw data.) Since multiple learning algorithms are available in the three approaches, it might be beneficial to use the same most effective learning algorithm as the meta-learner. That is, different learners are used at the leaves, but the same learner is used for the rest of the tree.

Moreover, we presently concentrate on improving learning speed via parallelism and prediction accuracy via diversity. However, the arbiter tree concept for parallel learning can be extended to improve accuracy as well. Recall that the arbiter tree has multiple levels of arbiters and the training set for an arbiter is derived from the arbiter subtrees. Currently, this training set consists of information derived from the original training data and/or the predictions from the arbiter subtrees. However, we can incorporate features (combinations of attributes) used in the arbiter subtrees into this set. In other words, at each level of arbiters, features are constructed from lower-level ones and are added to the training sets for the arbiters (i.e., performing *constructive induction* between levels). This way, more effective features would be in an arbiter's training set and hence a more accurate arbiter might be learned. Unfortunately, feature construction, requires non-negligible computation overhead, especially when large number of records are concerned, since a value has to be calculated for each new feature and record. However, this might be compensated for by reduced search time for some of the learning algorithms because of the presence of more useful features.

5 Concluding Remarks

The frameworks presented demonstrates that meta-learning can be used as a unified approach to facilitate multistrategy and parallel learning.

Various meta-learning strategies we have explored are also independent of the learning algorithms or parallel/distributed environment used. Preliminary empirical results suggest that certain strategies and meta-learners are more effective than others. Our results are preliminary and more experiments are being performed to ensure that the results we have achieved to date are indeed statistically significant, and to study how meta-learning scales with much larger data sets. The strategies discussed here are by no means final. We intend to refine our current strategies and explore others as our experiments progress.

Acknowledgements

This work has been partially supported by grants from New York State Science and Technology Foundation, Citicorp, and NSF CISE. We thank David Wolpert for many useful and insightful discussions that substantially improved the ideas presented in this paper.

References

- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth, Belmont, CA. 1984.
- Catlett, J. Megainduction: A test flight. In *Proc. Eighth Intl. Work. Mach. Learn.* 596-599. 1991.
- Chan, P. and Stolfo, S. Experiments on multi-strategy learning by meta-learning. Submitted to CIKM93. 1993a.
- Chan, P. and Stolfo, S. Toward multistrategy parallel learning in sequence analysis. In *Proc. First Intl. Conf. Intel. Sys. Mol. Biol.* To appear. 1993b.
- Chan, P. and Stolfo, S. Toward parallel and distributed learning by meta-learning. In

- Proc. AAAI Work. on Knowledge Discovery in Databases*. To appear. 1993c.
- Clark, P. and Niblett, T. The CN2 induction algorithm. *Mach. Learn.*, Vol. 3, 261–285. 1987.
- Cost, S. and Salzberg, S. A weighted nearest neighbor algorithm for learning with symbolic features. *Mach. Learn.*, Vol. 10, 57–78. 1993.
- Danyluk, A. Gemini: An integration of analytical and empirical learning. In *Proc. MSL-91*. 191–206. 1991.
- Drucker, H., Schapire, R., and Simard, P. Boosting performance in neural networks. *Intl. J. Pat. Recog. Art. Intel.* To appear. 1993.
- Flann, N. and Dietterich, T. A study of explanation-based methods for inductive learning. *Mach. Learn.*, Vol. 4, 187–266. 1989.
- Freund, Y. Boosting a weak learning algorithm by majority. In *Proc. 3rd Work. Comp. Learning Theory*. 202–216. 1990.
- Holder, L. Selection of learning methods using an adaptive model of knowledge utility. In *Proc. MSL-91*. 247–254. 1991.
- Mitchell, T. M. The need for biases in learning generalizations. Tech. Rep. CBM-TR-117, Dept. Comp. Sci., Rutgers Univ. 1980.
- Qian, N. and Sejnowski, T. Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, Vol. 202, 865–884. 1988.
- Quinlan, J. R. Induction over large data bases. Tech. Rep. STAN-CS-79-739, Comp. Sci. Dept., Stanford Univ. 1979.
- Quinlan, J. R. Induction of decision trees. *Mach. Learn.*, Vol. 1, 81–106. 1986.
- Schapire, R. The strength of weak learnability. *Mach. Learn.*, Vol. 5, 197–226. 1990.
- Silver, B., Frawley, W., Iba, G., Vittal, J., and Bradford, K. ILS: A framework for multi-paradigmatic learning. In *Proc. Seventh Intl. Conf. Mach. Learn.* 348–356. 1990.
- Stolfo, S., Galil, Z., McKeown, K., and Mills, R. Speech recognition in parallel. In *Proc. Speech Nat. Lang. Work. DARPA*. 353–373. 1989.
- Towell, G., Shavlik, J., and Noordewier, M. Refinement of approximate domain theories by knowledge-based neural networks. In *Proc. AAAI-90*. 861–866. 1990.
- Wah, B. et al. High performance computing and communications for grand challenge applications: Computer vision, speech and natural language processing, and artificial intelligence. *IEEE Trans. Know. Data. Eng.*, Vol. 5, 138–154. 1993.
- Wirth, J. and Catlett, J. Experiments on the costs and benefits of windowing in ID3. In *Proc. Fifth Intl. Conf. Mach. Learn.* 87–99. 1988.
- Wolpert, D. Stacked generalization. *Neural Networks*, Vol. 5, 241–259. 1992.
- Zhang, X., Mckenna, M., Mesirov, J., and Waltz, D. An efficient implementation of the backpropagation algorithm on the connection machine CM-2. Tech. Rep. RL89-1, Thinking Machines Corp. 1989.
- Zhang, X., Mesirov, J., and Waltz, D. A hybrid system for protein secondary structure prediction. *J. Mol. Biol.*, Vol. 225, 1049–1063. 1992.

Conceptual Clustering of Events Using Statistical Split Criteria

Bradley L. Whitehall

David J. Sirag, Jr.

United Technologies Research Center

East Hartford, CT 06108

{blw, djs}@antares.res.utc.com

Abstract

This paper describes a new approach to conceptual clustering called, clustering for single numeric attribute prediction (CSNAP). The events provided to the system have one attribute that is to be predicted using descriptions of the other attributes. The CSNAP system addresses the problem of constructing clusters of points that have a probabilistic value. In other words, the same state description does not necessarily have the same value for one of the key attributes. This paper describes how statistical methods are combined with machine learning techniques in order to build useful classifications of points.

Keywords: machine learning, multistrategy learning, conceptual clustering, prediction, statistics

1. Introduction

People have an incredible skill of observing a few instances of a situation and being able to learn from them. What is learned is not 100% accurate, but people still use the knowledge and develop a sense of its reliability. A simple example of such a task is learning when the lines are long at the bank. After just a few visits, people learn that going to the bank on Friday afternoons and before a holiday is to be avoided

because of long lines. When the bank patron visits the bank, lines are never exactly the same length, and there have been a few times on Friday afternoons when the lines were short. Yet, the general concept is learned and these rules about the bank are used with confidence. Problems like this occur everywhere: predicting the traffic rate at a certain time of day, determining what time of month the factory will be busy, etc.

While people are good at understanding and learning about fairly repetitive events, machine learning systems are not. The goal of this research is to produce a learning system that can understand such problems. In the process of learning to make accurate predictions there are two additional objectives:

- 1) The confidence in the prediction made for such problems can be stated along with the prediction.
- 2) The rules used to make the prediction are understandable to the humans using the system.

Statistical measures are able to provide the first capability. Symbolic learning systems are able to provide the second. This work integrates the strengths of each approach in order to provide capabilities not found in other learning systems.

The next section of the paper describes the motivation for developing the CSNAP system with special focus on why statistical and symbolic learning methods needed to be combined. The third section describes the CSNAP system and the fourth section presents results of an application where CSNAP has been used to predict building traffic patterns. The paper concludes with some final remarks on the system.

2. Motivation

The CSNAP system was developed for situations with the following criteria:

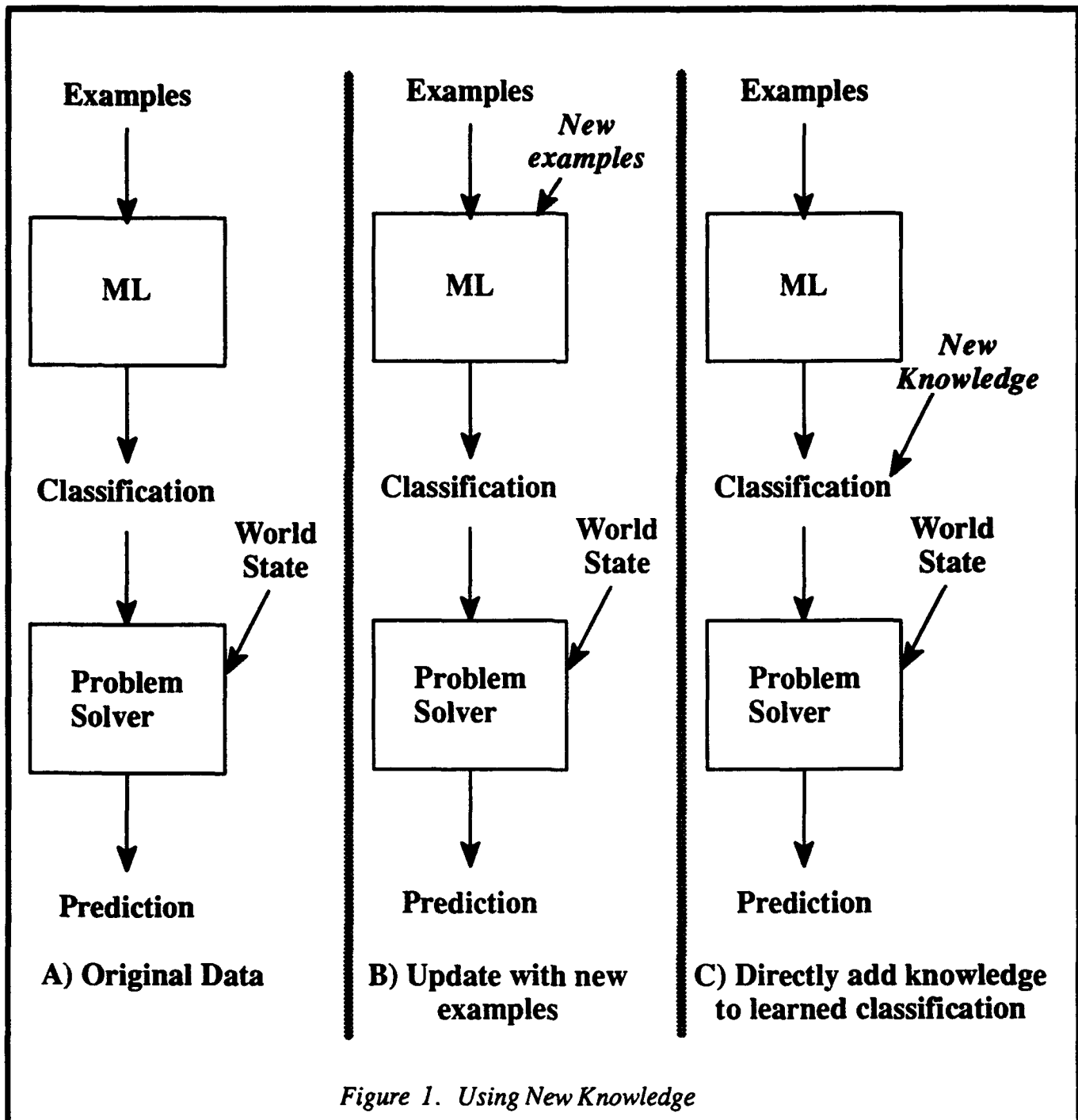
- ◆ A numeric (real valued) attribute is to be predicted (dependent variable).
- ◆ Two events with identical attribute values can have different dependent variable values.
- ◆ The attributes used to describe an event need not be independent.
- ◆ Accurate predictions for unseen events are required.
- ◆ Meaningful descriptions of event classes are produced.
- ◆ The knowledge produced from data should be (easily) extendible with additional human knowledge.

Many problems are defined by the criteria identified above. These problems define events that have a dependent variable that is influenced by the world in non-deterministic ways. Similar event descriptions do not guarantee that the dependent variable value will be identical. In order to gain knowledge about the world there must exist some regularity in those events. Statistics can capture this type of regularity by describing the mean and variance of the dependent variable values for a set of events. The mean value provides a "best guess" for events with that state description and the variance indicates the confidence or "goodness" of the guess. The goal of the CSNAP system is to cluster events to minimize the variance of the events within a class, thus improving the confidence of the predictions made using the classification.

Even the best learning system is limited by the data it is given. The learning system can capture the knowledge that is in the data, but not all relevant knowledge is necessarily in the data. This could occur if the problem solver using the knowledge produced by the learning system operates in a dynamic environment. For example, in the banking scenario presented earlier, the bank might send out a notice saying they are going to be closed next week to update their accounting system. That information is added to our knowledge about bank hours so that we will not go to the bank next week. That type of information also needs to be added to the knowledge structures produced from the learning system so the problem solver using it can account for the new knowledge. This

suggests the importance of being able to integrate knowledge from different sources. Figure 1 demonstrates this concept. Some learning systems deal with this by having the

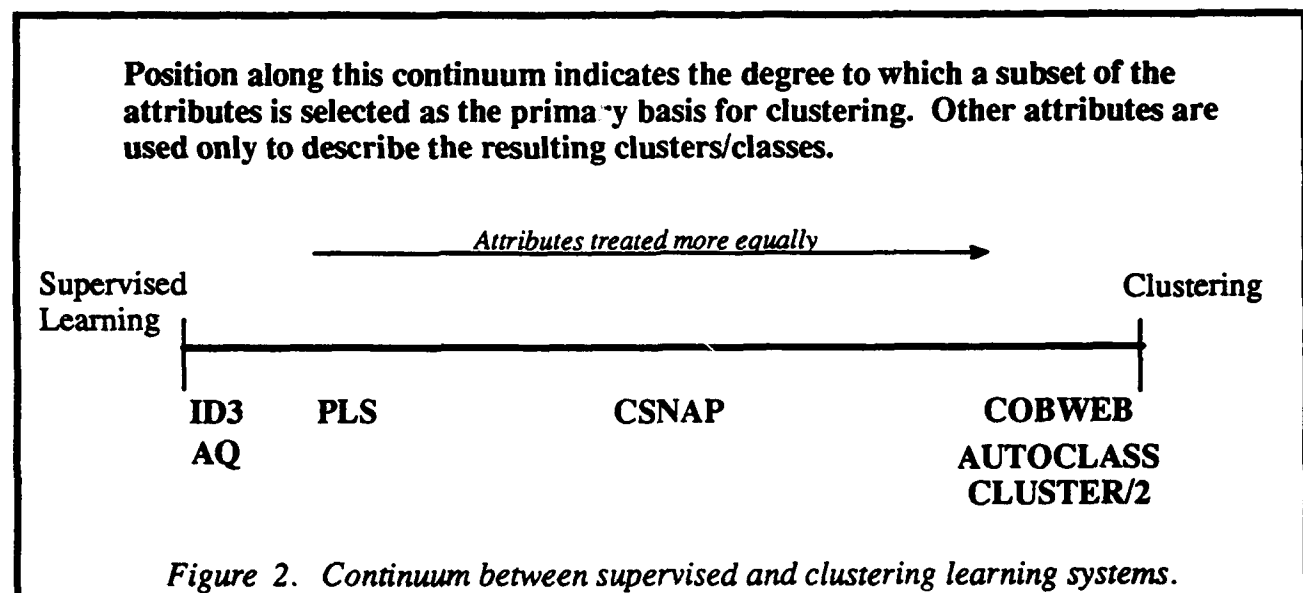
user generate new data with the new knowledge (Figure 1B). It is simpler and more reliable if the knowledge can be added directly to the systems knowledge structures (Figure 1C).



CSNAP is more like a conceptual clustering system (Michalski, 1983) than a classification system. Just because two events have similar rates does not (necessarily) mean that they belong together. As Figure 2 illustrates, there is a continuum between classification systems and clustering systems. Classification systems assume that one attribute identifies the class of an event. Pure clustering systems assume that all attributes are (nearly) equal in their importance to the found classes. CSNAP is more closely related to a goal directed clustering system (Stepp, 1986) because while the rate attribute is treated specially, it does not determine class membership by itself. In addition, CSNAP develops new clusters of the dependent variable values (new classes) in an attempt to perform prediction accurately.

Statistics can be used to induce a model from a set of data. For example, linear regression provides the best fit line to a set of points. Models produced with statistical techniques handle real values very well and often provide

a measure of confidence in the results. Unfortunately, the results are not easy to understand by the non-technician. It is also not clear how to add new knowledge to the model other than performing the steps outlined in Figure 1B. Several machine learning systems such as ID3 (Quinlan, 1986) and CART (Breiman, 1984) have used statistics or similar measures to induce a model. However, these systems used statistics to create the classifications but did not adjust those classifications in order to improve understandability of the resulting concept. Connectionist systems, though able to produce good classifications, do not provide the learned knowledge in a manner that is easily comprehended or extended by humans. CSNAP attempts to integrate the use of statistics with the desire to have understandable concepts. As explained in the next section, it is the struggle to provide results that are both highly accurate and comprehensible that makes CSNAP unique.



3. CSNAP Implementation

The basic CSNAP algorithm is presented in Figure 3. The essence of the approach is in steps 2 and 3, in which a kernel of events is selected and an initial description is formed. First the system splits the events classified at a node into two clusters of events in order to minimize the variance of the dependent attribute in the two clusters. The procedure to build the descriptions then uses these initial clusters as a guide to constructing the classes. Events are moved between the two clusters in order to construct a cohesive class. As points are moved, the variance of the clusters tends to increase, but the moves are selected to make the class descriptions more comprehensible. There is a tradeoff between the concise, clear descriptions and low variance. The user can adjust weights that are used to balance these conflicting goals.

The system performs a beam search on the space of possible class descriptions. Initially one class is given the description NIL, which covers all events. The description is extended by adding new attribute/value pairs (constraints) to the description in an attempt to cover one of the two initial clusters (the classes formed by minimizing the variance). Preference for adding attributes to a description are determined by the percentage difference of values between the two original clusters of examples. The percentage difference of values is computed by determining the percent of examples of the class that have a specific attribute value and getting the difference in this value for the two classes. Adding a previously unused attribute makes the description more specific and causes examples covered by the more general class to be moved to the other class for this split point. Adding a value to an attribute already used in the description makes the description more general.

CSNAP (points, current-node)

- 1) Sort points on dependent variable
- 2) Find the cluster of points with smallest projected variance
- 3) Build a description of the cluster
(Move points to/from cluster as needed to maintain a simple description)
- 4) Set new-points = points covered by description
- 5) Create new-node with new-points as child of current-node
- 6) Call CSNAP (new-points, new-node)
- 7) Set points = points - new-points
- 8) Call CSNAP (points, current-node)

Figure 3. CSNAP Algorithm

The descriptions are ranked for the search based upon a weighted sum of the following:

◆ **Variance**

- 1) The variance of the events covered by the description should be as small as possible.
- 2) The variance of the events not covered by the description should be as small as possible.

◆ **Generality**

The fraction of the description space covered by the description should be as large as possible.

◆ **Complexity**

The number of attributes and values used in the description should be as small as possible.

◆ **Number of examples covered**

- 1) The number of examples covered that were in one of the original clusters should be as large as possible.
- 2) The number of examples covered that were in the other original cluster should be as small as possible.

Each of these values is normalized and scaled to be between 0 and 1. The weights for each value can be adjusted to reflect user preferences on the system's results.

The outlined approach is similar to ID3 (Quinlan, 1986) from the standpoint of splitting off points, but similar to AQ (Michalski, 1975) in the way that descriptions are constructed. It

is a clustering system because the class memberships are determined dynamically as the clusters are built.

It should be noted that step 2 of Figure 3 requires the calculation of "projected variance." Projected variance is closely related to traditional variance but is based on a statistical estimate of how large the variance of the cluster might become as the number of samples increases. This penalizes small clusters because adding new samples can radically change their variance. If traditional variance were used clusters of size one would always have the best score. Effectively, the system calculates a tradeoff of the variance to the number of points included in the set. This allows the system to return larger classes with slightly worse variance over smaller classes with better variance. In addition, the system requires that a minimal number of examples be included in a class. This provides a parameter that allows the user to control the noise tolerated.

While discovering the classes, CSNAP builds a classification tree. The tree is organized with the root covering all the event descriptions and its branches covering more specific subsets. This insures that all possible event descriptions are covered by some node. Although binary splits are used in finding the minimal variance for the classifications, the trees produced are not binary. The found class with the smallest variance becomes a child of the other class, which remains as the current node. The child is then processed to determine if other splits can be found and likewise, the current node is also

processed again to determine if other splits may occur. Thus, a node can have any number of children, as long as each split produces an improvement in the variance of the data.

The classification trees produced by CSNAP provide the capability to answer a query about the dependent variable for a specific state (time), determine the state with the predicted optimal dependent attribute value, and incorporate user's knowledge by adding nodes.

4. Learning Traffic Patterns

In this section, results of running CSNAP on a model of building traffic are presented. This model is not used to help form classes, only to generate the training events. The dependent variable is the traffic rate of a building, the number of people entering and exiting the

building. The examples from the model are described with eleven attributes (shown in Figure 4) and the dependent variable rate. The model takes into account a number of factors when determining the rate including season and time of day. When the model creates a data point, it produces the typical traffic rate for the specified time period. The actual number of passengers observed is generated by sampling a distribution determined by the traffic rate supplied by the model. It is this non-deterministic number of passengers that is used by CSNAP for learning. This captures the idea that in real buildings the rate will not be the same every day at the same time; there exist natural fluctuations in the data.

Two months of data, sampling the traffic every five minutes, produced over 17,000 examples

Attributes	Type	Values
rate	real	≥ 0
seconds	circular	0-60 (integer)
minutes	circular	0-60 (integer)
hour	circular	midnight, 1am, 2am, . . . , 11pm
day-or-night	nominal	day, night
day-of-week	circular	mon, tue, . . . , sun
day-type	nominal	weekend, weekday
day	circular	1-31 (integer)
week	circular	first, second, third, fourth, fifth
month	circular	jan, feb, . . . , dec
year	linear	≥ 1980
season	circular	winter, spring, summer, autumn

Figure 4. Building Model Attributes

for training. The test set was produced by running the model over a (simulated) one week period (five minute sampling) immediately following the (simulated) two month training period. Results are shown in Figure 5 comparing CSNAP, ID3, and a 10-point moving average output for both the actual data point, and the true model value. The actual data section compares the systems' predictions to the observed traffic rate. The model section compares the predicted rates with the underlying model of the simulator. In the figure, AAE is the average absolute error of the value from the learned model, Var is the variance of those values, and Max is the maximum value difference (maximum error in the test set). For the ID3 results, the rates for the training examples were rounded to the nearest integer,

and each integer rate defines a unique class. As can be seen, CSNAP performs much better than the other approaches. Figure 6 graphically shows the data for a single day with the true model, the actual data, and the traffic predicted by the CSNAP model. This figure illustrates that CSNAP has learned a pattern very similar to the original model, based only on noisy samples drawn from that model, the actual data. CSNAP did not require a manually developed model of the environment, a description of numbers or types of classes to be found, or any theoretical assurances that the attributes are independent. Even so, CSNAP was able to create an accurate and fairly easy to understand/modify empirical model of the underlying process.

Actual Data			
	AAE	Var	Max
CSNAP	1.04	3.47	30.55
ID3	1.46	14.55	47.00
Mov Ave.	1.64	13.12	35.60
Model			
	AAE	Var	Max
CSNAP	0.57	1.89	24.95
ID3	1.31	14.56	47.17
Mov Ave.	1.51	12.74	29.60

Figure 5. Results on Traffic Data

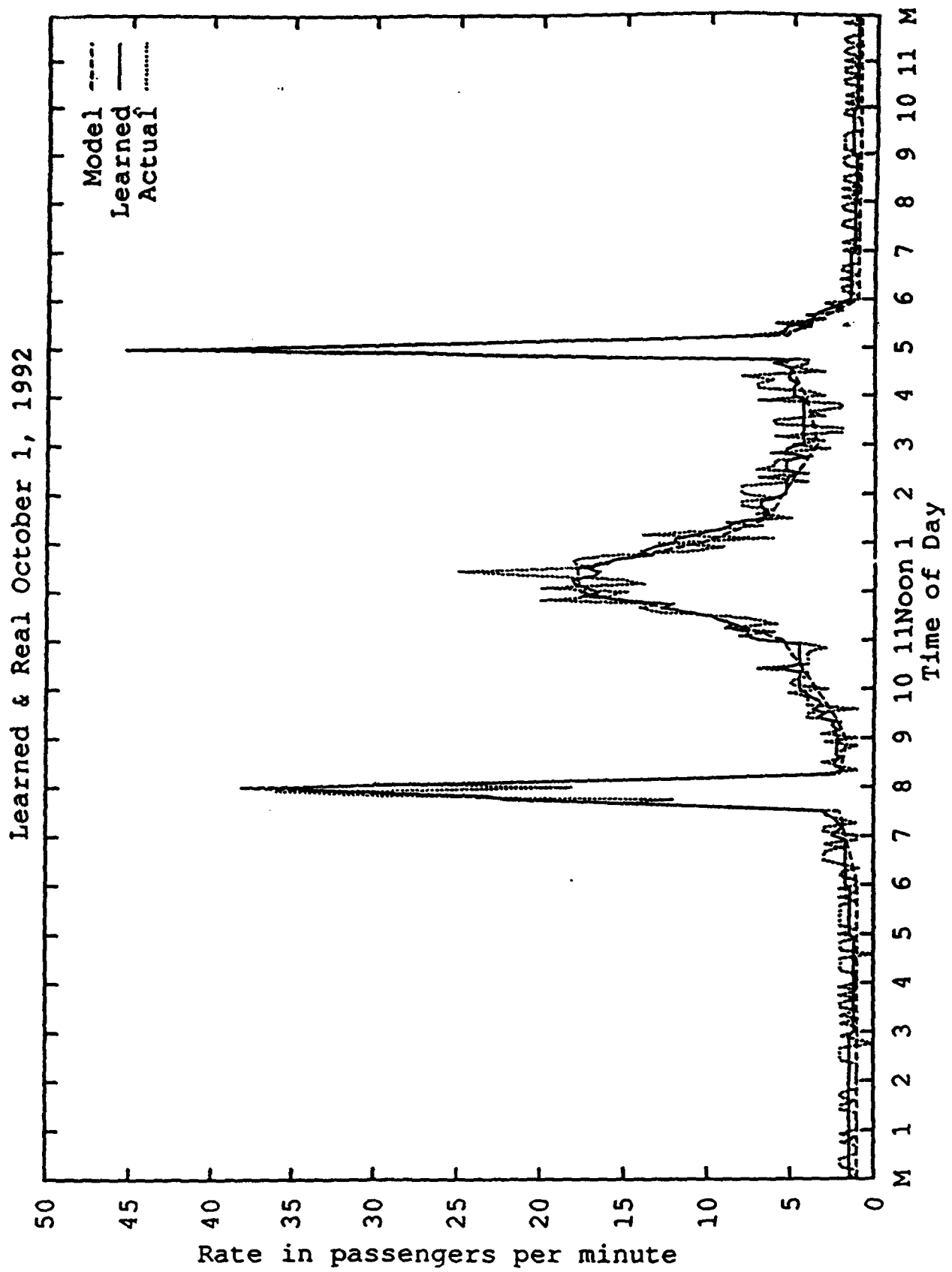
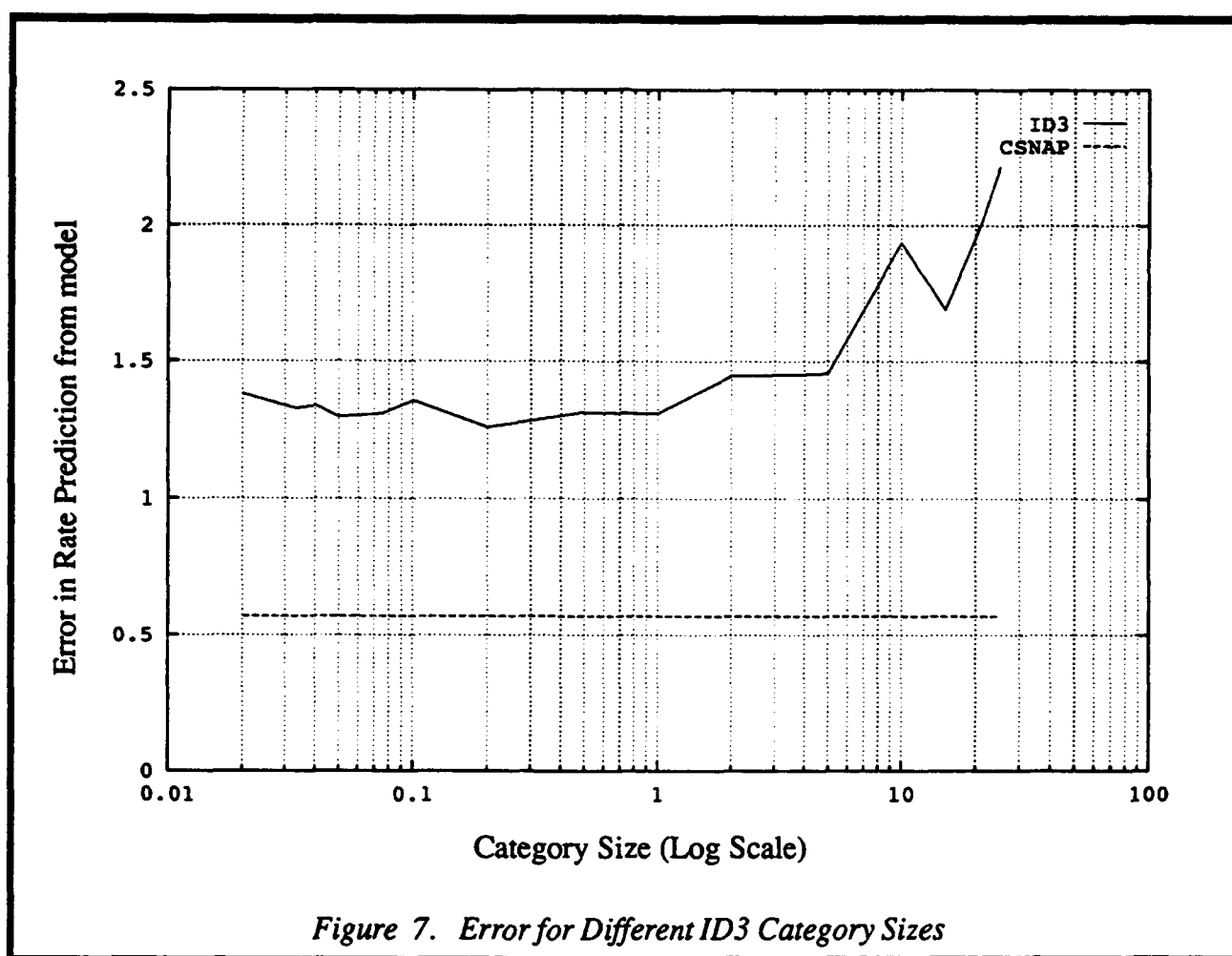


Figure 6. Data Plots for One Day

Since ID3 was not developed to handle real valued dependent variables, the bin size of 1 (integers) used in the previous results might not be the most appropriate. Figure 7 shows a plot of the errors for a variety of different bin sizes for ID3. The original bin size was 1, that is the actual rates were rounded to integers and placed in a class. As shown in the graph, this choice was near optimum for the binning size. Even with much smaller bin sizes (0.02) ID3 does not perform significantly better and the error rate is still much higher than that of the CSNAP approach. This helps to demonstrate that the problems CSNAP was designed to

address are not pure learning from example tasks. In addition, CSNAP helps to eliminate the step of manually choosing the correct bin size – CSNAP automatically identifies useful dependent value clusters (bins).

Figure 8 shows how the average error of the predictions decreases as CSNAP constructs the classification tree. The solid line is the average error over the one week testing period in passengers per minute. The dashed line is the average error for the tree plus one standard deviation of the error (equivalent to an error bar of one standard deviation for the average error line). CSNAP was designed to construct classes



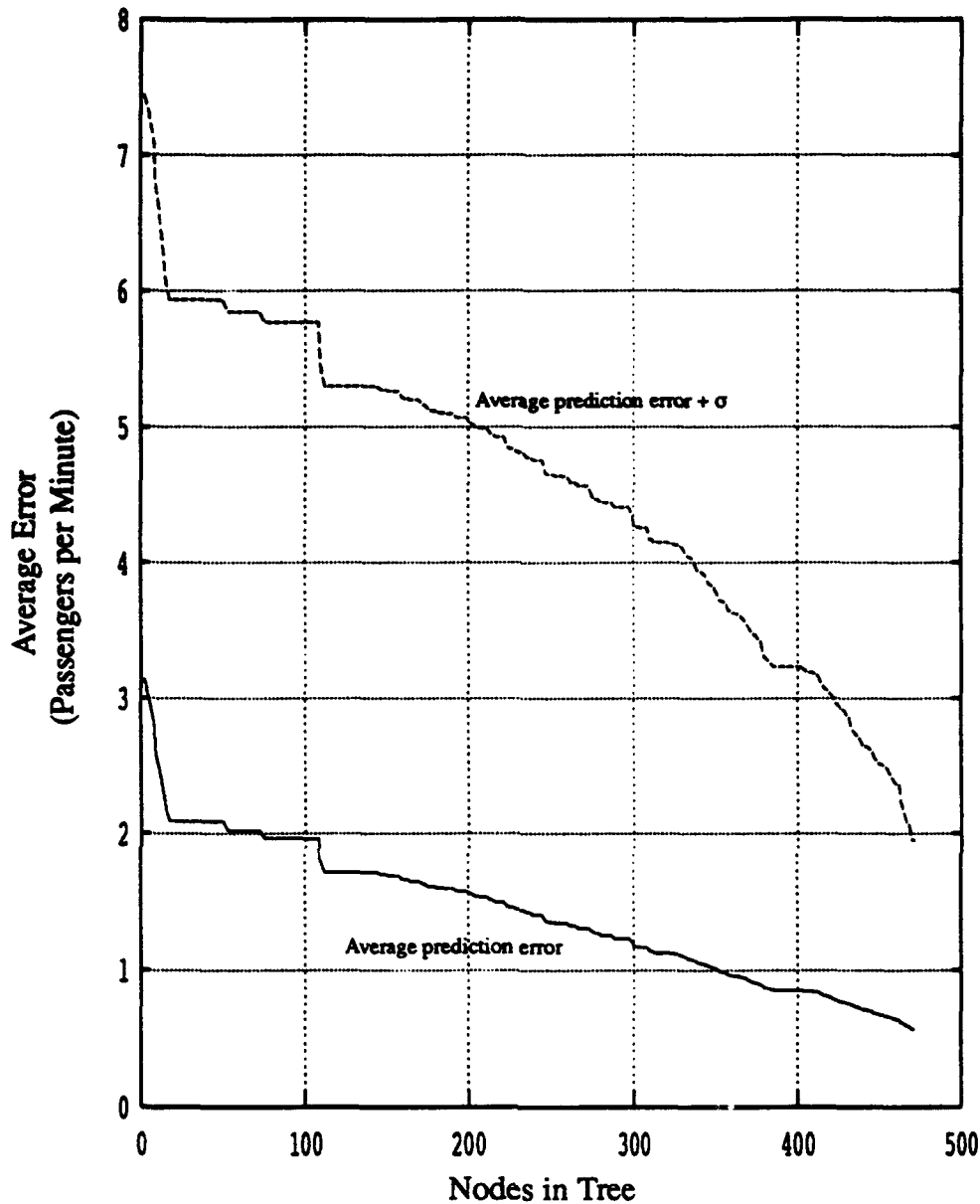


Figure 8. Error and uncertainty decrease as tree grows.

that have small variance and this graph indicates CSNAP is accomplishing that goal. By taking the difference between the two lines, it can be seen that the standard deviation of error decreases from 4.2 to 1.3 passengers per minute. This graph also indicates that the CSNAP system does not over learn from the training data. If it had, the error would increase as the tree grew. The completed tree has 470 nodes.

While this might not be as small as desired, the ID3 tree for this problem had over 67,000 nodes.^[1] This graph (Figure 8) was produced by testing the CSNAP tree after each node was added as it worked on the training data set. No pruning was performed.

[1]. A standard version of ID3 with CHI squared pruning was used.

Part of a classification tree produced by CSNAP is shown in Figure 9. The tree shown was generated with 4000 examples from the above training set. Each node of the tree is indicated on a separate line, with children being indented to the right under the parent. The first number in each line is a line number to be used for reference. The | symbol is used just to assist in lining up the nodes. The description of the nodes is placed between [and]. In this example all the descriptions are a single attribute but conjunctions of attributes are allowed. Internal disjunction of the attribute values is allowed as indicated by ... (see line 18). The first number following the description is the average value of the dependent attribute for all examples currently classified by this node. The second number is the variance for these examples and the third number is the number of examples currently classified by this node. CSNAP attempts to push events as far down in the tree as possible when classifying new points. For example, if an event comes in that has DAY-OF-WEEK = Wednesday and HOUR = 9am and a prediction is wanted, the system would place the event in the *first* child node that can match (cover) the event description. In this example, that is the node on line 4. Since Wednesdays are not covered by any of this node's children, the prediction would be 3.0 passengers per minute (on average with a variance of 1.87 passengers per minute based on a sample of 24 events). If instead the event had been a Friday, the prediction would have been 2.37 (line 5). Had the event in question occurred on a Saturday, the

prediction would have been 0.19 passengers per minute (line 2). It should be noted that it is possible for a child of a node to have a larger variance than its parent when the tree is finally constructed. When the node was originally separated from its parent the node's variance must have been smaller but since that time other examples could have been split from the parent leaving it with a better overall variance. Only the final examples left in a node are used to calculate its prediction (average).

It should also be observed from this tree segment that adding knowledge to the tree is trivial. For example, if knowledge were known that the traffic rate would increase at 2pm on Thursday, this knowledge could be added as a node after the root. As long as the information is added in the tree before the "standard" node used for that prediction, the new knowledge will override the learned predictions.

5. Conclusion

Adding the powerful features of statistics to symbolic machine learning systems is important. Using statistics, symbolic systems can add a confidence factor to their results. This confidence factor is important to those using the knowledge produced by the learning system. So important that some people will not use a system without them. Statistics must be integrated into the system so that the advantages of symbolic learning, such as comprehensible descriptions are not sacrificed. Unless the knowledge produced does not need to be understood by humans, it must be a form that they can

```

1  [] 17.20 33.21 20
2  | [(DAY-TYPE = WEEKEND )] 0.19 0.17 1440
3  | [(DAY-OR-NIGHT = NIGHT )] 1.49 0.32 1560
4  | [(HOOR = 9AM )] 3.00 1.87 24
5  | | [(DAY-OF-WEEK = FRI )] 2.37 1.01 24
6  | | [(DAY-OF-WEEK = THU )] 2.67 1.58 24
7  | | [(DAY-OF-WEEK = MON )] 2.37 1.62 24
8  | | [(DAY-OF-WEEK = TUE )] 2.92 1.84 24
9  | [(HOOR = 3PM )] 4.08 2.46 24
10 | | [(DAY-OF-WEEK = TUE )] 3.96 2.29 24
11 | | [(DAY-OF-WEEK = FRI )] 4.17 2.38 24
12 | | [(DAY-OF-WEEK = MON )] 4.42 3.01 24
13 | | [(DAY-OF-WEEK = WED )] 3.87 2.25 24
14 | [(HOOR = 10AM )] 0.00 0.00 0
15 | | [(DAY-OF-WEEK = THU )] 5.04 3.76 24
16 | | [(DAY-OF-WEEK = WED )] 4.62 3.78 24
17 | | [(DAY-OF-WEEK = MON )] 4.62 3.96 24
18 | | [(MINUTES = TWENTIETH-MIN...TENTH-MIN)] 5.12 3.86 24
19 | | | [(DAY-OF-WEEK = TUE )] 5.46 3.90 24
20 | [(HOOR = 2PM )] 5.54 4.72 24
21 | | [(DAY-OF-WEEK = TUE )] 4.87 2.97 24
22 | | [(DAY-OF-WEEK = FRI )] 4.83 4.47 24
23 | | [(DAY-OF-WEEK = THU )] 5.17 4.09 24
24 | | [(DAY-OF-WEEK = WED )] 5.12 4.82 24
25 | [(HOOR = 7AM )] 26.15 53.49 20
26 | | [(MINUTES = TENTH-MIN )] 2.50 0.91 20
27 | | [(MINUTES = ZERCTH-MIN )] 2.45 1.00 20
28 | | [(MINUTES = THIRTIETH-MIN )] 2.70 1.13 20
29 | | [(MINUTES = TWENTIETH-MIN )] 2.20 1.16 20
30 | | [(MINUTES = FORTIETH-MIN )] 13.55 28.98 20
31 | [(HOOR = 5PM )] 31.50 295.18 8
32 | | [(MINUTES = FIFTIETH-MIN )] 2.40 0.98 20
33 | | [(DAY-OF-WEEK = FRI )] 13.50 221.43 20
34 | | [(DAY-OF-WEEK = TUE )] 14.25 236.78 20
35 | | [(DAY-OF-WEEK = MON )] 14.30 251.19 20
36 | | [(MINUTES = THIRTIETH-MIN...FORTIETH-MIN)] 3.62 3.57 8
37 | | | [(MINUTES = THIRTIETH-MIN )] 4.25 4.22 8
38 | | [(MINUTES = TWENTIETH-MIN )] 5.62 6.79 8
39 | | [(DAY-OF-WEEK = WED )] 21.50 169.46 8
40 | [(HOOR = 8AM )] 39.30 138.77 20
41 | | [(MINUTES = TWENTIETH-MIN )] 2.65 0.71 20
42 | | [(MINUTES = FIFTIETH-MIN )] 2.15 0.80 20
43 | | [(MINUTES = THIRTIETH-MIN )] 2.15 0.90 20
44 | | [(MINUTES = FORTIETH-MIN )] 2.35 1.06 20
45 | | [(MINUTES = TENTH-MIN )] 13.75 19.31 20
46 | [(HOOR = 1PM )] 7.17 11.86 24
47 | | [(DAY-OF-WEEK = TUE )] 6.96 7.19 24
48 | | [(DAY-OF-WEEK = FRI )] 8.54 12.04 24
49 | | [(DAY-OF-WEEK = THU )] 7.54 7.43 24
50 | | [(DAY-OF-WEEK = WED )] 8.50 12.10 24

```

Figure 9. Segment of CSNAP tree

comprehend and extend as they deem necessary. CSNAP is an attempt to use statistics in this way. Although it has been shown that useful results can be obtained by this approach there are a number of areas for future work. In the future we expect to expand our comparison of CSNAP's results to results produced by other algorithms such as CART, AQ, Cobweb, and Autoclass. This task is difficult because direct comparisons between different algorithms, intended for different purposes, are often misleading. In addition, it is often difficult to find standard implementations of these algorithms that include all the features/extensions need for good performance.

In the future we also expect to expand CSNAP to identify more complex relationships in the data. Currently, CSNAP is essentially searching for regions of the space where the dependent attribute value is constant. Other, more complex relationships, such as a linear dependence on a single attribute, should also be considered. This not only could provide more reasonable generalization, but more understandable descriptions as well.

Acknowledgements

We would like to thank the members of the Artificial Intelligence group of UTRC for their support to this work.

References

- Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., *Classification and Regression Trees*, Wadsworth, 1984.
- Michalski, R.S., "Synthesis of Optimal and Quasi-Optimal Variable Valued Logic Formulas," *Proceedings of the 1975 International Symposium on Multiple-Valued Logic*, pp. 76-87, 1975.
- Michalski, R.S., Stepp, R.E., "Learning from Observation: Conceptual Clustering," In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach*, pp. 331-364, Tioga Publishing Co., 1983.
- Quinlan, J. R., "Induction of Decision Trees," *Machine Learning*, 1(1), 1986.
- Stepp, R.E., Michalski, R.S., "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structure Objects," In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach, Volume II*, pp. 471-498, Morgan Kaufmann Publishers, Inc., 1986.

Cooperation of Data-driven and Model-based Induction Methods for Relational Learning

Edgar Sommer, GMD*

Abstract

Inductive learning in relational domains has been shown to be intractable in general. Many approaches to this task have been suggested nevertheless; all in some way restrict the hypothesis space searched. They can be roughly divided into two groups: data-driven, where the restriction is encoded into the algorithm, and model-based, where the restrictions are made more or less explicit with some form of declarative bias. This paper describes INCY, an inductive learner that seeks to combine aspects of both approaches. INCY is initially data-driven, using examples and background knowledge to put forth and specialize hypotheses based on the "connectivity" of the data at hand. It is model-driven in that hypotheses are abstracted into *rule models*, which are used both for control decisions in the data-driven phase and for model-guided induction.

Key Words: Inductive learning in relational domains, cooperation of data-driven and model-guided methods, implicit and declarative bias.

1 Introduction

Inductive learning in relational domains has been shown to be intractable in general

[Kietz 93]. Many approaches to this task have been suggested nevertheless; all in some way restrict the hypothesis space searched. This is done first by restricting the language in which examples and background knowledge may be expressed. Additionally, the language in which the hypotheses are expressed is restricted. Data-driven methods, such as FOIL [Quinlan 90] and GOLEM [Muggleton/Feng 90], encode this restriction into the algorithm. In Model-based systems such as RDT [Kietz/Wrobel 92], GRENDL [Cohen 92] and CLINT [Raedt 91], the restrictions are made more or less explicit with some form of declarative bias¹.

Despite the challenge, learning relational concepts has great practical relevance, as evidenced by the ESPRIT projects *Machine Learning Toolbox* (MLT) and *Inductive Logic Programming* (ILP) funded by the European Community². The MOBAL system for building knowledge based applications [Morik 91][Morik et al. 93] has been used to develop several complex and practice-oriented applications. Experience has shown that both data-driven and model-based approaches offer advantages in such a knowledge engineering context [Sommer et al.]. Specifically, FOIL showed great speed in passing over the data discussed there, but the resulting rules were not satisfactory in coverage

*German National Research Center for Computer Science, AI Division (I3.KI), P.O.Box 1316, D-5205 St. Augustin 1, Germany, email eddi@gmdzi.gmd.de

¹For a comparison of GOLEM, RDT and CLINT see [Sutlic 92]

²P2145 and P6020 respectively. The work reported on here is funded in part through the latter.

of the goal concept. More significantly, FOIL (and the class of data-driven methods it represents) is a black box whose behavior cannot be modified. Some of its results may be statistically correct, but do not fit the design goals for the knowledge base, and their discovery inhibits the search for more "sensible" alternative rules. RDT, which is part of MOBAL, on the other hand was slower, but the results showed better coverage. More significantly, the declarative bias RDT uses can be tailored incrementally, so that later results were both better in coverage than FOIL's and comparable in speed. The work reported on this paper is an attempt at combining the speed of heuristically guided data-driven methods with the variability of model-based approaches.

2 INCY

A rule model is a higher order expression similar to a rule, except that predicate variables appear at the place of predicates (refer to [Kietz/Wrobel 92] for a precise definition). A rule is so seen as being an instance of a corresponding rule model, where the model's predicate variables are instantiated with specific predicate names. A model based learner such as RDT generates hypotheses by performing these instantiations in a systematic manner with predicates from the domain at hand³.

Rule models can and do exist independently of a specific domain (cf. "cliques" in [Silverstein/Pazzani 91]). Naturally, such generic models will not fit arbitrary data. INCY is learning algorithm designed to make a somewhat frivolous pass through a given domain and generate example hypotheses and rule models based on the connectivity of the

³The use of rule models as declarative bias goes back to [Emde 87]; a very similar approach is used in [Silverstein/Pazzani 91].

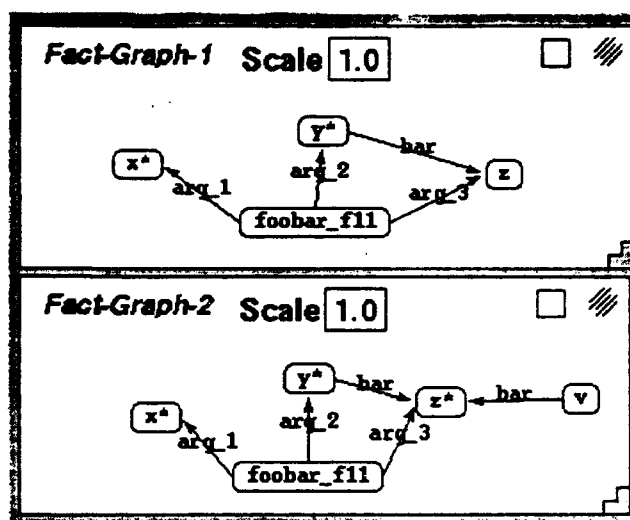


Figure 1: *Fact graphs before and after expanding object z*

specific data at hand. Even if none of the hypotheses are acceptable, or INCY's rules do not cover a sufficiently large subset of the known examples of the learning goal, a host of rule models are produced that directly reflect the structure of the data wrt to connectivity, i.e., which arguments of which predicates appear at which places in which other predicates.

2.1 Connectivity

The motivation for INCY's inner workings comes from a form of data inspection provided by MOBAL: the fact graph shows an incrementally extendable excerpt of a given knowledge base as a graph. Facts' arguments (the objects in the domain) are nodes, and the predicates in which they appear are arcs. Given an example of the learning goal in a domain, the graph initially shows all the facts known about the example's arguments, with the arcs linking arguments to other objects. The graph can be expanded to show all facts about these other objects, causing yet other objects to appear, etc. (Fig. 1). Eventually,

the graph will be fully expanded, displaying all facts about the example's arguments and objects they are linked to. Some conjunction of a subset of these facts must be a valid rule about the concept. Unfortunately, the number of conjunctions theoretically possible is quite large; often all facts in the domain are in some way 'connected' to the objects appearing in the example, so that any member of the powerset of the set of all facts in the domain is a possible hypothesis. The idea is to incrementally expand the set of candidates, guided by the incrementally expanding graph. If we have a learning goal $\text{foobar}(x,y,z)$ and the partial hypothesis

$\text{foo}(x) \ \& \ \text{foo}(y) \rightarrow \text{foobar}(x,y,z)$

we will want to add something about z to the premise before testing the validity of the hypothesis. The candidate must be among the facts about z :

$\text{foobar}(x,y,z)$

$\text{bar}(v,z)$

$\text{bar}(y,z) \dots$

The question is: which of these is the best new conjunct? Quinlan's FOIL [Quinlan 90] algorithm, for instance, uses the "information gain" heuristic to select a candidate, i.e. its value is measured in terms of the role it plays in *all* known examples of the goal. The INDUCE algorithms, though they are applied to structured objects rather than relational concepts, can be seen as relying on a pre-classification of candidates with additional heuristics [Michalski 83]. INCY takes a more basic approach, relying only on the connectivity information represented by the graph. It tries to select a candidate which is "most linked" to the other objects in the rule, and which preferably does not introduce a new object. In other words, it tries not to expand the graph, but rather find a fact concerning objects already visible. In the example, INCY would give preference to $\text{bar}(y,z)$, because $\text{bar}(v,z)$ would introduce a new object v (Fig. 1).

Vere's Thoth-pb induction method for relational productions uses association chains to selectively augment examples with background information before generalization [Vere 77]. Rather than logical induction, the topic here is finding operators that describe change in discrete scenes, similar to string rewrite rules and STRIPS operators. But the idea of selecting descriptors along relational links between objects and being conservative about expanding the association chain of such links is related to the approach taken here.

2.2 The INCY algorithm

INCY begins by selecting an example of the concept to be learned. For each object appearing as an argument in the example, the set of known facts about it is collected (the about set). One candidate from each of these sets is selected to form a preliminary premise for the hypothesis. More than one of the objects appearing in the example may occur in one candidate fact, so that the resulting candidate set may be of size smaller than the arity of the example. INCY keeps track of the candidates used across the iterations of this selection of preliminary conjuncts for one example to be able to avoid specializing permutations of the same hypothesis.

Linked-enough analysis

This premise is subjected to a "linked-enough" analysis before actually testing the hypothesis. In the course of this analysis, the premise may be augmented by one or more conjuncts to ensure that all arguments are sufficiently linked. An argument is linked-enough if it is not free, i.e. it appears in at least two literals of the hypothesis. It is linked *to* the other arguments of the literals it occurs in. INCY collects the 'problem' arguments that do not pass the linked-enough test and for each tries first to find a new conjunct linking it to the head arguments.

INCY top level

```

while there are uncovered examples of the goal concept
  ▷ select an example
  ▷ construct about sets for the example's args
  ▷ while there are combinations of candidates
    ▷ form prelim. conjunction by selecting one candidate fact from each about set
    ▷ perform linked-enough test/modifications on this conjunction of facts
    ▷ form & test hypo
    ▷ fail (backtrack) or specialize hypo
  ▷ end{hypo generation loop for one example}
  ▷ fail (backtrack to another example or end)
end{example selection loop}

```

INCY specialize hypo

```

while premise is not longer than c*arity(goal)
  ▷ construct about sets for all 'variables' occurring in hypo
  ▷ select a new conjunct for the premise from the about sets
  ▷ perform linked-enough test/modifications
  ▷ form & test hypo
  ▷ fail (backtrack) or further specialize hypo
end{specialization}

```

Figure 2: Pseudocode for INCY's data-driven phase

to other premise arguments otherwise (here giving preference to the other problem arguments). As a last resort, an argument may be marked as a constant if no suitable conjunct is found. Section 4 gives a formal definition. The linked-enough test/modification, in contrast to specialization described below, does not introduce new arguments into the hypothesis. It can be interpreted as a sort of consolidation phase after which all arguments are sufficiently bound (or described, or linked): either they appear repeatedly in the hypothesis, or they are marked as constants.

Hypothesis formation and test

The 'hypothesis' thus reached is a conjunction of facts – it is now turned into a true hypothesis by systematically substituting variables for constants. Testing procedures as described in [Kietz/Wrobel 92] are applied to this hypothesis. In this process, a rule model

is abstracted from the hypothesis (in analogy to abstracting a rule from a conjunction of facts, here the predicate names are turned into variables). The results of the test can be one of:

```

rule known
rule model known
hypothesis too specific
hypothesis accepted
hypothesis too general

```

In the first four cases, INCY backtracks. This is straight-forward in the hypothesis too specific case, but why not specialize when rule known, rule model known or hypothesis accepted? The decision not to specialize here is what underlies the built-in frivolity of INCY's data-driven phase and allows it to put forth a maximum of structurally different hypotheses in relatively short time, since it avoids producing structurally equivalent hypotheses. In one strat-

egy of cooperation discussed (Section 3), a model-based learning phase is initiated when hypothesis accepted, so that such hypotheses are systematically enumerated. Note that a rule model is retained for possible later use regardless of whether the corresponding hypothesis was accepted.

If, in backtracking, there are no other ways of ensuring the hypothesis is linked-enough, other preliminary candidates are selected. If no good combinations (as described at the beginning of this section) are available, a new example is selected. Only in the final test result case (hypothesis too general) is the hypothesis is specialized.

Specialization

Specialization is a modified version of the linked-enough modification above: the new conjunct should ideally link a head variable to one of the premise variables. If no such fact is available, the next preferred type is one that links two premise variables (both may introduce a new variable in addition to linking two existing ones). If this, too, is impossible, a new conjunct binding one of the non-head variables is selected. The crucial difference to the linked-enough modification is that here new variables may be introduced into the premise. A subsequent call to the linked-enough analysis ensures that none of these remain free. From here on, INCY proceeds in the same manner as above (see Fig. 2). The depth-bound for specialization is a function of the goal concept's arity $c \cdot \text{arity}(\text{goal-concept})$. c can be modified via parameter.

3 Cooperation

The two basic forms of cooperation between data-driven and model-based methods have already been touched on:

1. A rule model is abstracted from a hypothesis and used to test hypotheses. This call

structure influences INCY's behavior in several ways:

- After abstracting a hypothesis to a *rule model*, the new model is compared to existing models by an extension of theta-subsumption [Kietz/Wrobel 92]. Redundant, i.e. structurally equivalent models are caught at this point, and this causes INCY to backtrack (search for a different specialization or select different conjuncts for the preliminary hypothesis).
 - If the data-driven phase should discover a *rule* that is already known, this is discovered during the test and specialization is aborted.
 - The criteria used to decide if a *hypothesis* is acceptable, too specific or too general are parametrized. Changing their values cause INCY to accept more or less general and more or less bold hypotheses. Since INCY's decisions about when to specialize, when to select different conjuncts about the current example's arguments, and when to try a different example depend on the outcome of this test, changing their values significantly change INCY's behavior.
2. The rule models generated during INCY's pass over data can be used in a subsequent, more stringent analysis. During this model-based pass, the models are instantiated with fitting predicates from the knowledge base [Kietz/Wrobel 92]. This may result in two different types of rules:
- Rules *structurally equivalent* to one of those discovered during the data-driven phase: each of those initial rules can be understood as being an example for a set of rules. Naturally, such structurally equivalent rules will, in general, be quite different in "meaning".
 - Rules not structurally equivalent to any of the rules discovered by INCY, but based on one of INCY's rule models nevertheless. INCY misses many acceptable rules because its pass is guided not only by the

test results that pertain to a specific hypothesis, but also by the (pre-) existence of rule models. This is the reason for INCY's built-in frivolity, but also for its speed. INCY may discard a hypothesis whose corresponding rule model - with different instantiations of the predicate variables - yields plausible rules. These rules are found in the model-based pass.

3.1 Other Schemes of Cooperation

Sequential cooperation

The basic cooperation scheme discussed above is dynamic in that INCY makes model generation and hypothesis testing calls, but learning itself is strictly sequential: INCY preprocesses data - discovering some rules and more rule models - and a model-guided induction step can be called subsequently. In the knowledge engineering context provided by MOBAL, this is fitting, as it allows the use of INCY for a quick initial analysis of data and RDT for a more concerted later effort.

A more selective use of the rule models produced by INCY is also possible, however. After INCY has completed its pass, for instance, a domain expert may scan the rules found and select those that seem most promising; RDT can be made to search only for similar rules by passing on only those models which the promising rules are instances of. Alternatively, the expert may inspect the rule models found by INCY and pass only those deemed plausible on to RDT.

Dynamic cooperation

In the same vein, INCY may make selective calls to a model-based pass from *within* its data-driven pass over data. For instance, before backtracking or specializing when a hypothesis is accepted, a model-based pass may be initiated to learn with the corresponding rule model only; or such a pass may be initiated with the list of models that were suc-

cessful in this sense *after* one of the data-driven passes inner loops has terminated. A third possibility is to call model-based methods with the most successful branch of the model subsumption tree: the set of increasingly special (by theta subsumption) models corresponding to the highest number of rules discovered by INCY.

The first of these dynamic cooperation schemes has been implemented. Note, however, that this significantly alters INCY's behavior. The sequential version described in Section 2.2 is designed to quickly discover a maximum of vari-structured rules, each of which can be viewed as an example for a class of structurally equivalent rules. On the other hand, the *dynamic* combination of model-based and data-driven strategies will test the entire classes of rules immediately, and take accordingly longer to complete a pass. The sequential version will be of more use in a knowledge engineering context where a quick preliminary analysis of large amounts of data is desirable. In the dynamic version, INCY functions more as a model-generator for an RDT-like method, producing the models it needs on the fly, based on the connectivity of the knowledge base at hand.

4 Results

The hypothesis language defined procedurally by INCY can be described as:

$$\begin{aligned} \mathcal{L}_{HS} = & \{ h = L_{prems} \rightarrow l_{concl} \mid \\ & \exists \sigma : \{ l_{concl} \cup L_{prems} \} \sigma \subseteq \mathbf{Bg} \\ & \wedge \text{linked-enough}(\{ l_{concl}, L_{prems} \}) \\ & \wedge |L_{prems}| \leq \text{depth-bound}(l_{concl}) \} \end{aligned}$$

where l_{concl} is the conclusion literal (the head), L_{prems} is the set of premise literals, and \mathbf{Bg} is the set of facts that make up the knowledge base⁴. σ is a substitution of terms

⁴Note that instances of the learning goal are also in \mathbf{Bg} , i.e. INCY may learn recursive rules.

for variables such that two different variables are replaced by different terms:

$$\sigma \in \{v_i/t_i \mid \forall 1 \leq i \leq j \leq n : \\ v_i \neq v_j \Rightarrow t_i \neq t_j \\ \wedge t_i, t_j \text{ are ground}\}$$

The space of such hypotheses is searched top-down. Note that INCY's data-driven phase discovers only a subset $\mathcal{R}_{frivolous} \subset \mathcal{R}_{valid} \subset \mathcal{L}_{HS}$ of all such rules valid in a given domain. The model-based phase continues this work, but does find all \mathcal{R}_{valid} because specialization is aborted when a rule model is already known (recall Section 2.2). The next restriction imposed on the base language (function-free Horn clauses with negation) is that the premise together with the goal of a hypothesis must be linked-enough (Section 2.2):

$$\begin{aligned} \text{linked-enough}(\{l_{concl}, L_{prems}\}) &\iff \\ \forall v \in V_{l_{concl}} \exists v_{prems} \in V_{L_{prems}} : v &= v_{prems} \\ \wedge \forall v \in V_{L_{prems}} \exists v_{concl} \in V_{l_{concl}} : & \\ \text{linked}(v, v_{concl}) & \\ \wedge \forall v \in V_{L_{prems}} \exists l_1, l_2 \in L_{prems} : & \\ l_1 \neq l_2 \wedge v \in V_{l_1} \wedge v \in V_{l_2} & \end{aligned}$$

where $V_{l_{concl}}$ are the head variables, and $V_{L_{prems}}$ are the variables occurring only in the premise. The linked relation between variables of a hypothesis is best defined recursively:

$$\begin{aligned} \text{linked}(v_1, v_2) &\iff \\ \exists l \in \{l_{concl} \cup L_{prems}\} : v_1, v_2 \in V_l & \\ \vee \exists v_3 : \text{linked}(v_1, v_3) \wedge \text{linked}(v_2, v_3) & \end{aligned}$$

This formalization defines a superset of the hypotheses that pass the linked-enough test in the top-level loop (Fig. 2): the modifications made during linked-enough analysis there do not introduce new variables. These are introduced during specialization, so that the definition above is precise wrt INCY's overall behavior.

The main effect of this is that hypotheses with free variables are not put forth by INCY.

Note that variables for which this condition cannot be fulfilled are turned into constants during linked-enough analysis. On the whole, this is a weaker restriction than that of *ij-determinacy* used in GOLEM, since determinacy is not supposed, and there is no explicit depth limit (i) for the variables occurring in a hypothesis, so that INCY is able to learn in domains where GOLEM isn't. FOIL's information gain heuristic ensures that only linked conjuncts are added to the premise during specialization, but accepts hypotheses with free variables, so that its implicitly defined language cannot be compared directly to \mathcal{L}_{HS} . Some tests indicate that FOIL's heuristic does not do well with sparse data because there is little information gain to work with. In the presence of a few negative examples, FOIL can no longer apply the closed world assumption and finds non-generative rules which do not cover any of the examples⁵. INCY does not make the closed-world assumption, so that its results are not affected.

The main difference between FOIL and INCY, as far as results are concerned, is that the rules discovered are quite different structurally, and that INCY discovers far more rules. This may or may not be deemed an advantage, as INCY's rules tend to be more redundant than FOIL's (several rules cover an example). Among these there are often more sensible ones – from a knowledge engineering point of view – than those FOIL homes in on. When using INCY as a model-generator for induction algorithms using declarative bias, such as RDT, GRENDL and CLINT, diversity in models is a plus. INCY learns recursive rules and rules with constants, and hence models which reflect this. FOIL was more affected by sparsity of examples, where information gain has little to work with, and

⁵In these rules, not all variables were bound in the premise. MOBAL's inference engine does not apply such faulty rules.

rules induced using INCY's models scaled up better when new examples were incorporated into the knowledge base, but experiments in other domains are underway to corroborate this.

Acknowledgements

This work was partially funded by the European Community in ESPRIT project "Machine Learning Toolbox" (MLT) and is partially funded by ESPRIT project "Inductive Logic Programming" (ILP). I thank Werner Emde, Joerg-Uwe Kietz, Katharina Morik, Stefan Wrobel and the other members of the ML groups at GMD and Dortmund University for valuable discussions.

References

- [Cohen 92] William Cohen. Compiling Prior Knowledge into an Explicit Bias. In Derek Sleeman and Peter Edwards (eds.), *Proc. ML92*, pp. 102-110, 1992.
- [Emde 87] Werner Emde. Non-Cumulative Learning in METAXA.3. In *Proc. 10th International Joint Conference on Artificial Intelligence*, pp. 208-210, Los Altos, CA, August 1987. Morgan Kaufman.
- [Kietz 93] Jörg-Uwe Kietz. Some Lower Bounds for the Computational Complexity of Inductive Logic Programming. In *Proceedings of European Conference of ML 1993*, 1993.
- [Kietz/Wrobel 92] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models. In Stephen Muggleton (ed.), *Inductive Logic Programming*, chapter 16. Academic Press, London, 1992.
- [Michalski 83] Ryszard S. Michalski. A Theory and Methodology of Inductive Learning. In *Machine Learning — An Artificial Intelligence Approach*, volume I, pp. 83-134. Morgan Kaufman. Los Altos, CA, 1983.
- [Morik et al. 93] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications*. Academic Press, London, 1993. to appear.
- [Morik 91] Katharina Morik. Balanced Cooperative Modeling. In Ryszard Michalski and Gheorghe Tecuci (eds.), *Proc. Workshop on Multistrategy Learning*, Harpers Ferry, 1991. also to appear in the Machine Learning Journal.
- [Muggleton/Feng 90] Stephen Muggleton and Cao Feng. Efficient Induction of Logic Programs. In *Proc. First Conf. on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha Publishers.
- [Quinlan 90] J.R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5(3):239 - 266, 1990.
- [Raedt 91] Luc de Raedt. *Interactive Concept-Learning*. PhD thesis, Kath. Univ. Leuven, Leuven, Belgium, February 1991.
- [Silverstein/Pazzani 91] Glenn Silverstein and Michael Pazzani. Relational Cliches: Constraining Constructive Induction During Relational Learning. In Birnbaum and Collins (eds.), *Procs. of the Eighth International Workshop on Machine Learning*, pp. 203-207, San Mateo, CA, 1991. Morgan Kaufmann.
- [Sommer et al.] E. Sommer, K. Morik, J.-M. Andre, and M. Uszynski. What online Machine Learning can do for Knowledge Acquisition. submitted.
- [Sutlic 92] Irma Sutlic. Restricting the search space in inductive logic programming systems GOLEM, MOBAL and CLINT. In *Proc. ISSEK Workshop '92*, 1992.
- [Vere 77] Steven A. Vere. Induction of relational productions in the presence of background information. In *Proc. of the 5th International Joint Conference on Artificial Intelligence*, pp. 349-355, 1977.

Multistrategy Constructive Induction: AQ17-MCI

E. Bloedorn, R.S. Michalski and J. Wnek

Artificial Intelligence Center
George Mason University
Fairfax, VA 22030
{bloedorn, michalski, wnek}@aic.gmu.edu

Abstract

This paper presents a method for multistrategy constructive induction that integrates two inferential learning strategies—empirical induction and deduction, and two computational methods—data-driven and hypothesis-driven. The method generates inductive hypotheses in an iteratively modified representation space. The operators modifying the representation space are classified into "constructors," which expand the space (by generating additional attributes) and "destructors" which contract the space (by removing low relevance attributes or abstracting attribute values). Constructors generate new dimensions (attributes) by analyzing original or transformed examples (data-driven) and by analyzing the rules obtained in the previous iteration (hypothesis-driven). Destructors detect the irrelevant components of the representation space by rule-based inference or statistical analysis. The method has been implemented in the AQ17-MCI program. The preliminary results from applying it to a problem with noisy training data and large number of irrelevant attributes demonstrated a superiority of the method over other constructive induction methods both in terms of the predictive accuracy, as well as the overall simplicity of the generated descriptions.

Key words: multistrategy learning, inductive inference, constructive induction, representation space, concept learning.

1. Introduction

Conventional concept learning techniques generate hypotheses in the same representation space in which original training examples are presented. In many learning problems, however, the original representation space is inadequate for formulating for the correct hypothesis. This inadequacy can be evidenced by a high degree of irregularity in the distribution of instances of the same class in the original representation space.

In a situation, there exists a mismatch between the complexity of concept boundaries in the space and the capabilities of the descriptive constructs of the representation language to describe the boundaries. Consequently, if the boundaries are highly irregular, typical constructs used in learning systems will likely be inadequate for representing them. Such typical constructs include nested axis-parallel hyper-rectangles (decision trees), arbitrary axis-parallel hyper-rectangles (conjunctive rules with internal disjunction, as used in VL1), hyperplanes or higher degree surfaces (neural nets), compositions of elementary structures (grammars), etc.

To address such problems, the idea of constructive induction has been introduced (Michalski, 1978; Watanabe and Elio, 1987,

Matheus and Rendell, 1989; Rendell and Seshu, 1990; Wnek and Michalski, 1991). Constructive induction can be viewed as a "double-search" process, that searches both for a hypothesis and for an adequate representation space in which to express this hypothesis.

Most constructive induction methods use a specific technique within one basic computational method. Basic methods are classified to data-driven, hypothesis-driven and knowledge-driven (Wnek and Michalski, 1991, 1993). Recently, there has been a trend toward "multistrategy" constructive induction approaches that integrate several techniques and methods.

This paper presents early results on the development of a multistrategy constructive induction system, AQ17-MCI, that aims at integrating a wide range of constructive induction techniques and methods. The basic ideas and the architecture of the system are based on the Inferential Theory of Learning (ITL), proposed by (Michalski, 1992). In ITL, learning is viewed as a "goal-directed process of modifying the learner's knowledge by exploring the learner's experience."

As mentioned above, a constructive induction learner performs two types of searches—a search for an inductive hypothesis and a search for an adequate representation space in which the hypothesis is represented. These two types of searches require different types of search operators.

The search for a hypothesis applies operators provided by the given inductive learning method. For example, the AQ17-MCI method (briefly, MCI) uses operators employed in the AQ-type learning systems, such as "dropping

conditions," "extension against," "adding an alternative," "closing interval," and "climbing a generalization tree."

The representation space search operators modify the representation space. The AQ17-MCI method uses both "constructors," that expand the space by adding new dimensions (attributes), "destructors" that contract the space by removing less relevant attributes and/or abstracting values of some attributes.

To perform a representation space search, meta-operators are introduced that allow the system to suggest different representation space search operators and methods ("constructive induction strategies"). Using the ITL framework, the selection of constructive induction strategies is done by applying the operator selection rules based on the evaluation of hypotheses generated in consecutive iterations i.e. by "exploring the learner's experience".

This paper describes several techniques and methods for representation space search, their integration in AQ17-MCI system, and the results from testing the system and comparing it with several other systems. The hypothesis space search is assumed to be done by the standard AQ-type algorithm.

2. Related Research

The MCI method is relevant to both the research in constructive induction and multistrategy learning. Related work includes the system LAIR (Watanabe and Elio, 1987), and "Principled Constructive Induction" (Mehra, Rendell and Benjamin, 1989). LAIR uses domain-specific background knowledge to construct new attributes. Principled

constructive induction uses geometric interpretations of various constructors to guide their selection. Neither of these approaches, however, possesses the wide range of constructors and destructors available in MCI.

Other related systems are STAGGER that integrates techniques for Boolean, numerical and weight learning (Schlimmer, 1987). GABIL, for adaptive strategy selection, based on classification performance (Spears and Gordon, 1991), and MBAC, which uses parabolic models of strategy performance for strategy selection (Holder, 1991). The strategy selection in MCI also draws inspiration from research on the development of large scale inference systems, especially INLEN (Kaufman, Michalski and Kerschberg, 1991). In INLEN, knowledge acquisition or discovery is based on *learning* rules from expert-supplied examples. The automated acquisition of rules is most suitable to areas where expertise is difficult to quantify, or where rules may need to be modified often, such as in the case of strategy selection for constructive induction.

Several systems have been developed that exhibit constructive induction capabilities. Some of the earliest were INDUCE (Michalski, 1980) and LEX (Mitchell, Utgoff and Banerji, 1983). Many systems are based either on an analysis of the training data, i.e., data-driven systems (e.g., Schlimmer, 1987; Bloedorn and Michalski, 1991), or an analysis of hypotheses, i.e., hypothesis-driven (Matheus and Rendell, 1989), (Pagallo and Haussler, 1990), (Wnek and Michalski, 1991, 1993).

These techniques are not very useful in situations requiring different types of knowledge representation space change. For example, learning from complex and noisy sensory data (e.g., learning to recognize

textures or shapes), seems to require a number of different techniques for the representation space change.

Given several such techniques, a problem arises of choosing the one that is most fit for a given situation. This problem is somewhat analogous to the problem of choosing an inductive learning method to fit the given problem at hand. Aha (1992) has proposed to solve the latter problem by using meta-rules that link the properties of training datasets with various empirical inductive learning methods.

To choose among many representation space modification operators, the MCI method uses meta-rules that link the properties of the training datasets and properties of the hypotheses generated from these datasets with the appropriate representation space modification operators.

3. The MCI Method

3.1 An Overview

The MCI method integrates a large number of different representation space modification techniques that are used to determine an adequate representation space for concept learning. The process of concept learning itself if done by an AQ-type inductive learning method.

A general flow diagram for the MCI method is shown in Figure 1. The input data are initially a user-provided training dataset plus a characterization of the initial representation space, which includes a description of attributes, their types and their domains. The training dataset is split into a primary and a secondary dataset. The primary training set is inputted to the Decision Rule Generation

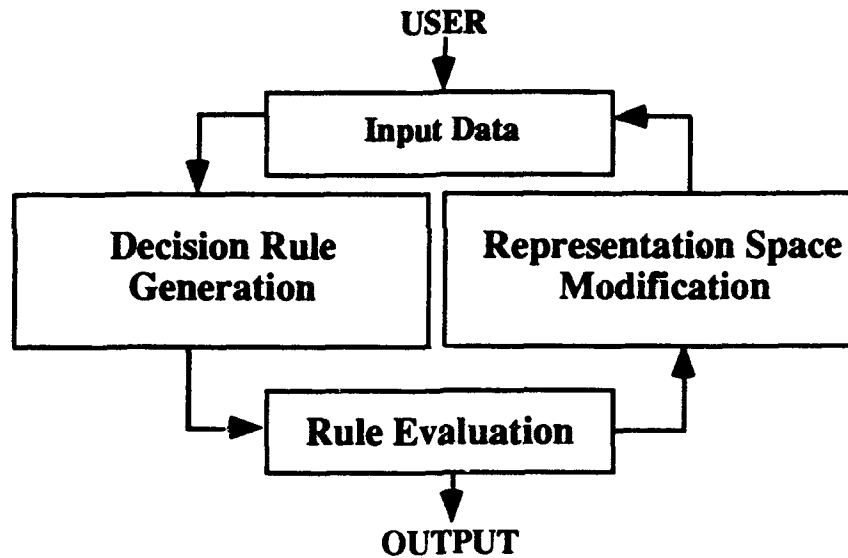


Figure 1. A functional diagram of the MCI method.

module, which uses an empirical inductive learning program (AQ14) to generate general concept descriptions (rulesets). The obtained rulesets are evaluated in terms of their complexity and their performance on the secondary training set. Based on the results of this evaluation, the system decides either to stop the learning process (the obtained rules are outputted as the solution), or to move to the Representation Space Modification module. This decision is based on special control meta-rules (Section 3.2.). The final decision rules are evaluated on the testing examples to determine their performance. Figure 2 shows the partitioning of the input examples into different classes (primary and secondary training examples, and testing examples), and explains how they are used.

The representation space modification is done by an application of various constructive induction operators, acting as constructors or destructors. Once a new representation space has been determined, both the primary and secondary training dataset is reformulated into this space, and the process is repeated. The

next sections describe in greater detail various aspects of the above process.

3.2 Determining When to Modify the Representation Space

The representation space needs to be modified if there exists a mismatch between the distribution of examples in the space and the capability of the representation language to adequately describe this distribution. This mismatch can be removed either by developing a learning algorithm capable of generating more complex discrimination surfaces in the given representation space, or by changing the representation so that simple discrimination surfaces will do the job. For some problems, the first approach is infeasible.

The constructive induction approach is to modify the representation space to remove the mismatch. An illustrative example of such a mismatch is the "bit parity detection" problem. A description of binary strings with this property in terms of the bit positions in the string is very complicated and long. If, however, one generates an additional attribute

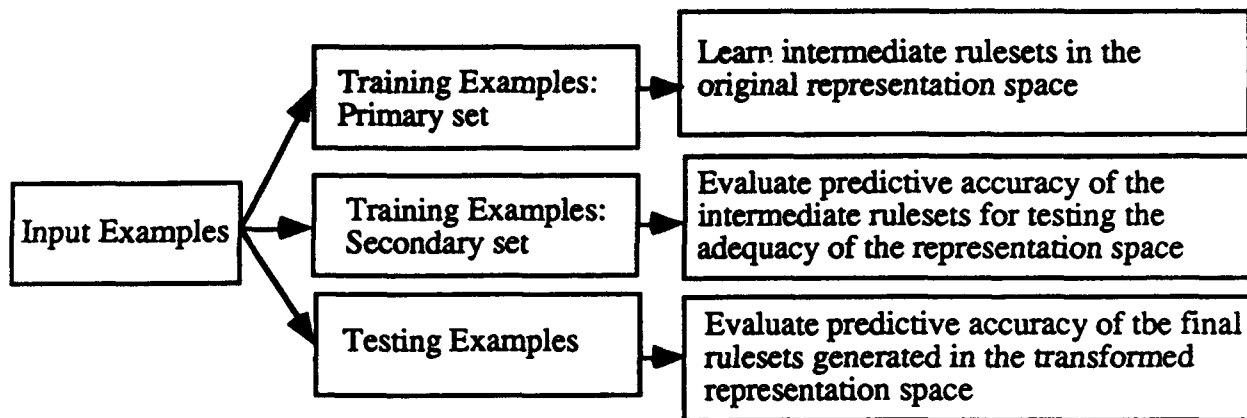


Figure 2. Subsets of the examples and their role.

(a dimension in the representation space) "mod2" of the sum of the bits in the string, the problem becomes trivial. The MCI approach is to apply a wide range of such operators for representation space change in order to determine a description space in which it would be easy to find the correct or approximately correct decision rules.

The problem arises of how to detect the need for representation space change. The MCI method solves this problem on the basis of the "quality" of descriptions (rulesets) generated by the Decision Rule Generation Module. The "quality" of the obtained ruleset is evaluated in terms of its predictive accuracy on the secondary training set and its complexity. If the quality is "satisfactory", according to the user or some heuristic criterion, then the process stops. A description of the dataset of examples in terms of certain meta-attributes is stored in the system's knowledge base to serve as a "meta-training example."

Meta-examples are used to represent datasets that both require some kind of representation space modification and those that do not. These meta-examples are used to develop meta-rules guiding the decisions about the need for the representation space change. If the rule quality

is unsatisfactory, the method enters the Representation Space Modification module.

3.3 Determining How to Modify the Representation Space

3.3.1 Meta-attributes and Meta-rules

The representation space is modified by applying a variety of operators. These operators include both constructors that expand the space and destructors that contract the space (see Section 3.3.3). The choice of the operators is guided by the meta-rules that relate the properties of the example dataset and the rule evaluation results on the secondary training set to the most appropriate operators. These rules are initially provided by the user, and later improved through learning from the meta-examples mentioned below.

The meta-examples are described in terms of meta-attributes. These meta-attributes are organized into four classes: those characterizing types of the original attributes (numeric, multivalued nominal, Boolean, etc.), those characterizing the attribute quality, such as the attribute *utility* (Imam and Michalski, 1993) or the entropy measure (Quinlan, 1983), those characterizing the expected level of quality of

Meta-attribute category	Meta-attribute	Values	Explanation
Meta-attributes detecting the presence of various types of attributes	Numeric_attributes_present	Yes, No	Yes, if data contains two more numeric attributes; No, otherwise
	Nominal_attributes_present	Yes, No	True, if data contains two or more multi-valued nominal attributes; False, otherwise
	Boolean_attributes_present	Yes, No	True, if data contains two or more Boolean attributes; False, otherwise
Meta-attributes characterizing the attribute quality	Irrelevant_attributes_present	Yes, No	Yes, if data contains any irrelevant attributes; No, otherwise
	Attribute_group_quality	Sufficient, Insufficient	Sufficient, if the minimum quality of the set of attributes is above an assumed threshold; Insufficient, otherwise
Meta-attributes estimating the quality of examples	Overprecision	Yes, No	Yes, if an attribute in the given set is measured with an excessive precision. No, otherwise
	Attribute_value_noise_level	Error rate in percentage (1..100%)	Teacher-estimated error rate in the measurement of the attribute values in the examples
	Classification_noise_level	Error rate in percentage (1..100%)	Teacher-estimated error rate in the assignment of examples to classes by the teacher ("mislabeling")
Meta-attributes estimating ruleset performance	Performance_estimation	Accuracy in percentage (1..100%)	Predictive accuracy of the last ruleset generated from the primary training example set and tested on the secondary testing set.
	Performance_change	Strongly Up, Up, No change, Down, Strongly Down	Measures the difference in performance between the n^{th} ruleset learned and the $n-1^{\text{st}}$ ruleset learned

Table 1. Meta-attributes for characterizing datasets.

the examples, and those characterizing the changes in the performance of the generated rules on the secondary training dataset. Table 1 presents a list of meta-attributes. With the exception of *Irrelevant_attributes_present*, and *Attribute_group_quality*, which can be automatically calculated in a manner described below, the values of these meta-attributes are provided by the user.

a) Attribute Type

The applicability of the representation space modification operators (for short, RSM operators) depends on the type of the attributes. For example, arithmetic operators apply to numeric attributes, logical operators apply to Boolean and multi-valued nominal attributes, etc. The type of attributes for which different RSM operators are available are currently numeric, multi-valued nominal and Boolean.

b) Attribute Quality

Attribute quality measures the ability of a single attribute to discriminate among given classes of examples. An attribute may contribute individually, or as part of an attribute group. Individual attribute quality can be measured statistically by calculating the ability of an attribute to partition the example set appropriately. One such measure is the information gain used in ID3 (Quinlan, 1983).

The value of the meta-attribute "*Attribute_group_quality*" is "True" if each attribute in the given group of attributes has gain greater than a user-defined minimum. This meta-attribute is useful for detecting situations in which each original attribute has some relevance, but not very high, which may be suggestive of the need for some multi-argument

representation space operator (e.g., a mod_x of the sum of the values of the attributes).

The contribution of an individual attribute in the context of a set of attributes can be measured by analyzing rules generated from examples described in terms of these attributes (Wnek and Michalski, 1993). The meta-attribute "*Irrelevant_attributes_present*" views an attribute as irrelevant if this attribute is not present in the rules, or is present only in the "light" rules (rules associated with low values of t-weight parameter the coverage of training examples by a rule).

An alternative measure of the individual attribute quality is the *attribute utility* (Imam and Michalski, 1993). An attribute utility is the sum of the class utilities of an attribute. The class utility of an attribute is the number of classes whose attribute value set has no common values with the value set occurring in the given class. An attribute is considered irrelevant if its attribute utility is low. Thus, *Irrelevant_attributes_present* is true if, for any attribute present in the data, the utility of that attribute is below threshold.

c) Example Quality

The quality of training examples is characterized in terms of three meta-attributes. The first one, "Overprecision," tests if a given attribute is measured with an excessive precision. In such a situation, the valueset of the attribute is reduced, and the values of this attribute in the examples are substituted by more abstract values. The second meta-attribute, "*Attribute_value_noise_level*" expresses a teacher-estimated error rate in the measurement of the attribute values in the examples. The third meta-attribute "*Classification_noise_level*" expresses a

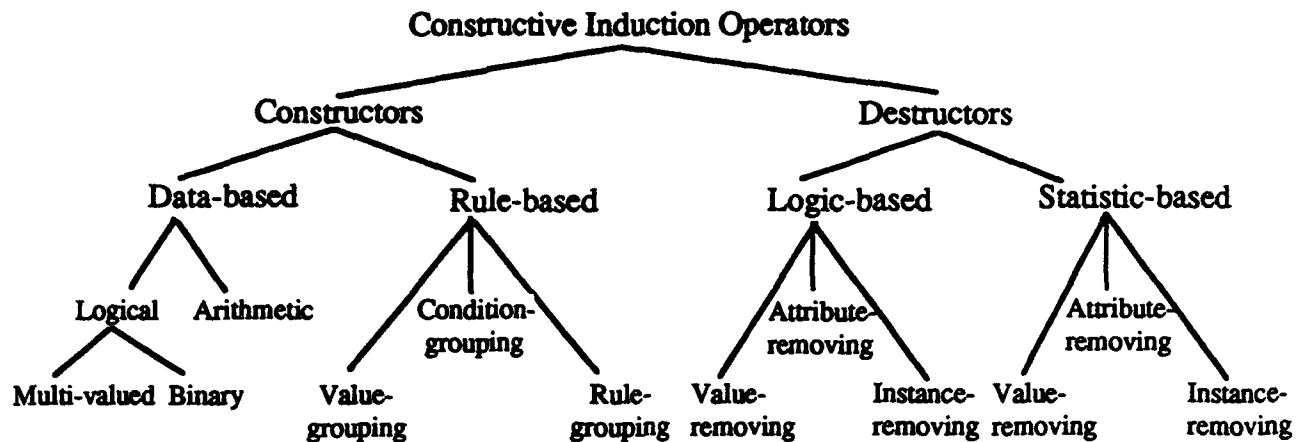


Figure 4. A hierarchy of constructive induction operators

teacher-estimated error rate in the assignment of classes to examples to classes by the teacher ("mislabeling").

Overprecision is reduced by proper quantization of the attributes (e.g., Kerber, 1992). Noise in the data is reduced by filtering training data through "heavy" rules (with high-weight) in the induced descriptions (Pachowicz, Bala and Zhang, 1992).

d) Rule Performance

There are two meta-attributes in this category: "Performance_level" that measure the performance accuracy of rules on secondary training examples, and "performance change" that expresses the change in performance from one rule generation iteration to the next. These meta-attributes help guide the selection of representation space modifiers by detecting when successive iterations are making significant positive increases in rule quality, or when the change in quality has declined or ceased.

3.3.2 Applying Meta-rules for Operator Selection

Each example dataset is characterized by a vector of the previously listed meta-attributes

and their values. Operator selection is a deductive process of applying previously learned representation space modification operator rules to these meta-attribute vectors. This matching procedure calculates a degree of match between the meta-example and the RSM rules using ATEST (Reinke, 1984). Representation space modifiers are then ranked in decreasing order of match. If no single RSM rule is the top rule, then the user is asked to select. This selection may be based on the user's preference for different types of modifications such as arithmetic constructions over logical constructions.

It may occur that the same RSM operator is repeatedly selected. In other words the search stagnates on a local maximum. MCI attempts to prevent this by updating the database characterization after each ruleset evaluation. Since the meta-attributes are updated continuously, the selection stage picks the operator that best matches the *current* database characterization. If all available operators fail to match the description (i.e., the degree of match is below a threshold) then selection stops and MCI evaluates the current ruleset on the testing examples. At minimum the best performance of MCI will be that which is achieved when no modifications are made to the representation

space. In this case the performance of MCI will be equal to just selective induction.

The set of constructive induction operators can be organized hierarchically as shown in Figure 4. This hierarchical organization captures the relationships between CI operators and allows selection rules to provide better guidance when confronted with new domains. The current MCI system has capabilities for both types of constructors, logical attribute and logical instance destructors, and statistical attribute-value removal.

The system was bootstrapped by providing meta-examples describing datasets for which appropriate representation space modifications were already determined. This was done to confirm if the resulting meta-rules agree with experience. Descriptions for seven domains were provided including: two monk's problems (Thrun, et al., 1991). Congressional voting records from 1984 (Bloedorn and Michalski, 1991), texture data (Pachowicz, et al., 1992), artificially generated DNF4 functions and multiplexer 11 (Wnek, 1993) and finally wind-bracing data from a civil engineering domain (Arciszewski et al., 1992).

The appropriate RSM operator for each domain was found experimentally. These meta-examples were given to AQ14 classified by CI method so that strategy selection rules could be learned. Table 2 shows the learned representation space modification operator selection rules. Default rules are used in the case of RSM operators that do not yet have meta-examples in the knowledge base.

The degree of match between an example and a rule is calculated using the method of ATEST (Reinke, 1984). The degree of match for all meta-rules matching to the dataset

characterization greater than threshold are displayed to the user.

dc1_numeric	←
	[Numeric_attributes_present = Yes]& [Attribute_value_noise_level = 0%]
dc1_boolean	←
	[Numeric_attributes_present = No]& [Nominal_attributes_present = No]& [Irrelevant_attributes_present = No]
dc1_nominal	←
	[Nominal_attributes_present = Yes] & [Attribute_value_noise_level = 0%]
hci_rule_grouping	←
	[Attribute_value_noise_level = 0%] [Irrelevant_attributes_present = Yes]
rule_based_instance_removal	←
	[Overprecision = Yes] & [Attribute_value_noise_level = 5%]
stat_based_attribute_value_removal	←
	[Overprecision = Yes] & [Attribute_value_noise_level = 5%]

Table 2. Examples of learned meta-rules for representation space modification

3.3.3 Example Reformulation

After the representation space modification has been selected, the training data are reformulated in this space. The generation module has a number of fundamental CI operators with which it can modify the primary and secondary training set. These operators include those used by a number of previous systems (Bloedorn and Michalski, 1991), (Pachowicz, et al., 1992), (Wnek and Michalski, 1993). Some of these fundamental operators have been reported

by others, notably Rendell and Seshu (1990). The following MCI operators are equivalent to the terms used in Rendell: attribute removal (projection), attribute-value removal (puncturing), and hypothesis-driven constructive induction (superpositioning).

a. Attribute Removal

Attribute removal makes a selection of a set X' of attributes from the original attribute set X . In MCI, a logic-based attribute removal is performed based on the quality of an attribute (as described by the meta-attribute "Irrelevant_attributes_present"). The irrelevancy of an attribute is calculated by analyzing rules generated by the Decision Rule Generation module. For each attribute, a sum is calculated of the total number of examples covered by a discriminant rule which includes that attribute. Attributes that are irrelevant will be useful only to explain instances that are distant from the majority of examples in the distribution. Thus, these attributes will have low total-weight sums. Logic-based attribute removal is performed in MCI by AQ17-HCI.

b. Attribute-value Modification

Attribute-value modification can be either the addition, (concretion) of values to an existing attribute domain, or the deletion (abstraction) of attribute values. Currently MCI implements only abstraction, based on the chi-square correlation between an attribute-value interval and the class. Using chi-square to quantize data was first proposed by Kerber (Kerber, 1992). Attribute value modification (AVM) selects a set $V' \subset V$ (where V is the domain of A) of allowable values for attribute A . AVM can be used to reduce multi-valued nominal domains, or real-valued continuous data into useful discrete, values. Discretization is especially

important for empirical induction methods that allow only small number of discrete attribute values such as ID3 (Quinlan, 1983) and AQ (Michalski, 1983a). In MCI, statistic-based attribute-value removal is performed by a chi-square based method.

c. Hypothesis-driven CI

Hypothesis-driven CI (HCI) is a method for constructing new attributes based on an analysis of inductive hypotheses. Useful concepts in the rules can be extracted and used to define new attributes. These new attributes are useful because explicitly express hidden relationships in the data. This method of hypothesis analysis as a means of constructing new attributes is detailed in a number of places including (Wnek, 1993; Wnek and Michalski, to appear 1993). Wnek and Michalski define a hierarchy of hypothesis patterns from the simplest (value-groupings) to the most complex (rule-groupings). which is implemented in AQ17-HCI. AQ17-HCI is used in MCI to perform rule-based constructions of attributes based on value-groupings, condition groupings and rule-groupings, and attribute removal (see section a).

d. Data-driven CI

Data-driven (DCI) methods build new attributes based on an analysis of the training data. One such method is AQ17-DCI (Bloedorn and Michalski, 1991). In AQ17-DCI new attributes are constructed based on a generate and test method using generic domain-independent arithmetic and boolean operators. In addition to simple binary application of arithmetic operators including $+$, $-$, $*$, and integer division, there are multi-argument functions such as maximum value, minimum value, average value, most-common value, least-

common value, and #VarEQ(x) (a cardinality function which counts the number of attributes in an instance that take the value x). Another multi-argument operator is the boolean counting operator. This operator takes a vector of m boolean-valued attributes ($m \geq 2$) and counts the number of true values for a particular instance. This approach is able to capture m -of- n type concepts. Data-based logical construction in MCI is performed by AQ17-DCI using the multi-argument functions of #VarEQ(x), most-common, least-common, boolean counting, and binary boolean operators. Data-based arithmetic construction is performed by AQ17-DCI through maximum, minimum, average, and +, -, * and integer division.

e. Instance Removal

Instance removal (IR) methods detect and filter noisy, or misclassified training examples. The method used in MCI is a logic-based approach implemented in AQ-NT (Pachowicz, et al., 1992). The IR operator removes instances from the training data if they are covered by 'light' disjuncts. Light disjuncts are those disjuncts in the rule which cover only a small fraction of the total number of instances in the class. Thus if the ratio of covered instances to total instances in a class is below some threshold the covered instances are removed from consideration by the training data. The relationship between the weight of learned rules and the plausible prototypicality of examples was first described in the AQ15-TRUNC method (Michalski, 1983b). Other work, based on calculating the statistical significance of individual instances is done in (Holte, Acker and Porter, 1989)

3.4 Rule Evaluation

Once a CI operator has been selected and applied to the data, or as a part of the initial

detection step, the resulting classification rules must be evaluated. (Figure 1). Control is returned to either the representation space modification module, or the process stops dependent upon rule quality. Rule evaluation is based on a number of criteria. As described in (Bergadano, et al., 1988) the quality of a concept description may be judged by three criteria: accuracy, simplicity and cost. In their approach, as in MCI, the user selects the relative importance of each of these criteria.

The *predictive accuracy* of a rule set is a measure of the ability of the rule set to correctly classify examples that were previously unseen. In MCI predictive accuracy is tested using a secondary training set. The secondary set is selected from the data the learner has not yet seen. Both primary and secondary data are not used for testing. Rules learned from the primary training set, but which perform well on the secondary set, are also less likely to be overfitted to the original data. Predictive accuracy is measured as the percentage of secondary training examples correctly classified.

Complexity of a ruleset is evaluated by counting the number of rules in the ruleset and the total number of conditions.

Cost is a measure of the price of evaluating the values of variables used in the description. Each variable has an associated cost provided by the user. A parameter within the rule-learning program, AQ, can be used to control the use of attributes in a description based on cost. For this reason cost is not included in the quality calculation presented here.

The final quality of the rule is evaluated lexicographically. Rulesets are evaluated first according to the accuracy criterion. If the

accuracy is within a user defined threshold of the goal accuracy, the ruleset is then further evaluated according to the complexity criterion. If, the ruleset does not meet the minimum standard for accuracy it is rejected and no further processing is done. The lexicographic evaluation permits the user to set a constraint on the minimum allowable accuracy.

3.5 Storing Experience of Operator Selection: Meta-examples

Each time a strategy is selected, and evaluated against the secondary training examples data, the results of the modification must be stored. If the application resulted in an improvement in rule quality, the meta-example characterizing the dataset is inserted into the knowledge base under the class representing RSM operator which made the useful modification of the representation space. If the quality remained constant or declined, the user determines if the meta-example should be stored. The problem of learning meta-rules which not only link a dataset to an operator, but also make the selection in the context of previous selections is discussed in section 6.

Given set of classified meta-examples, new meta-rules can be learned or improved. Meta-rules are generated by AQ14. The new meta-rules generalize the previous meta-examples. Meta-rules will now be capable of classifying unseen databases according their suitability to representation space modification. Examples of learned meta-rules are presented in Table 2.

4. Experiments

The MCI method was tested in an artificial problem, an extension of the difficult second monk's problem in which both misclassification noise and irrelevant attributes are added. This problem, "Noisy and Irrelevant Monk2" (NIM2), extends the difficulty of the second Monk's problem, by including 5% random misclassification noise (9 training examples) and 7 irrelevant, randomly generated attributes to the original set of 6. The goal concept of the NIM2 problem, like the original monk 2, is: "exactly two of the 6 attributes take their first value". In the monk 2 problem, dcinominal attribute construction modified the training data by adding a new attribute which represented the number of values which take their first value. This modification allowed AQ

Problem	Method	Accuracy (Exact match)	Complexity	
			#Rules	#Conds
Noisy Monk2 (5% noise) #Classes=2 #Attributes=13 AVD Size=3	AQ14 (No data modifications)	47.2 %	37	327
	AQ14 (with stat. attrib. value removal)	46.8 %	43	236
	AQ17-HCI (Rule-based attribute construction and removal)	42.1 %	13	55
	AQ-NT (Rule-based instance removal)	43.1 %	19	125
	AQ17-DCI (Data-driven attribute construction)	81.5 %	17	122
	MCI	90.2 %	8	23

AVD - "attribute value domain"

Table 3. A performance comparison of the MCI method with several single strategy methods.

to find the goal concept resulting in rules which perfectly stated the goal concept. AQ17-HCI also solved this problem by constructing new attributes based on "xor-rule-patterns" (Wnek, 1993).

The NIM2 problem, however, is more difficult. For this problem, dci-nominal construction builds the same attribute, but the goal concept has been disrupted by misclassified examples. This results in fairly accurate, but complex rules (81% predictive accuracy, 17 rules).

The MCI method was also applied to the problem. The detection step was performed with AQ14 generating 37 rules with a predictive accuracy of 47%. When presented with NIM2, MCI first invoked rule-based instance removal. Using AQ-NT 5% of the training examples were removed, and new rules were learned. There were 19 new rules with an accuracy of 43%. MCI next invoked dci-nominal. dci-nominal constructed a new attributed representing the number of attributes which take their first value. With this new attribute, AQ was able to generate 12 rules with an accuracy of 86%. When the operator selection module was invoked again, the meta-rule for dci-nominal construction continued to have the greatest match to the meta-example describing the dataset. When dci-nominal was invoked again, no new attributes were constructed-no representation space modification was made-so the next best method, HCI was selected by the user.

In this third representation modification, HCI added two new attributes and deleted seven features x2, x3, x6, x7, x8, x10 and x13. When AQ was invoked on the transformed database 8 rules with only 23 conditions were generated with an accuracy of 90%. MCI

selection ceased when dci-nominal was selected again, and no new attributes were constructed. The combination of rule-based instance removal (AQ-NT), Data-driven CI (dci-nominal) and Hypothesis driven CI and attribute removal, produced a ruleset which has significantly fewer total rules (8 vs. 17), significantly shorter rules (23 vs. 122 total conditions) and which are better performing (90% vs. 81%) than the next best single strategy constructive induction method of AQ17-DCI. In table 4, MCI is compared to AQ14 which has not methods for data modification, the results of AQ14 after processing the data with a chi-square based attribute value removal method, AQ17-HCI, AQ-NT and AQ17-DCI.

The problem of determining the context of operator selection decisions is a matter of future work. It is interesting to note that when different meta-rules are used (characteristic vs. discriminant), the MCI method selects only dci-nominal construction and then HCI. The resulting ruleset is still superior, in predictive accuracy to any single strategy method, but is more complex (88.2% accuracy, 17 rules, 57 conditions).

5. Summary

This paper presented a methodology of multistrategy constructive induction that integrates two inferential learning strategies-empirical induction and deduction, and two computational methods--data-driven and hypothesis-driven. Empirical induction was performed in the Rule Generation module, and in the search for appropriate Representation Space Modifications (the double search of constructive induction). Deduction was used in the application of learned meta-rules to the characterization of incoming datasets in order to

select an appropriate representation space modification. MCI includes "constructor" and "destructor" modifiers. Modifier selection is based on meta-rules learned from the results of past applications of modifiers.

The MCI approach was tested on a problem, the NIM2, characterized by misclassification noise and irrelevant attributes. The MCI method produced rules which surpassed not only traditional selective induction learning (no representational modifications), but also single strategy methods in terms of the quality of rules produced. The quality of the resulting ruleset was superior both in terms of predictive accuracy on the testing examples, and complexity.

6. Future Work

One important area of improvement of the current method is the determination of a good criterion when to stop applying representation space modification (RSM) operators. In general, rule quality changes when representation space modifications are made. Currently, RSM operator selection, application and evaluation is repeated until the user is satisfied with the current ruleset quality. But if the user is not satisfied, and the change in the rule quality has been negative, the question arises as to whether the system should not recommend to the user some new ways of continuing the search process.

Such a decision should be based on a new type of meta-knowledge that keeps track of which RSM operators have been tried so far, and which have not. The meta-attribute set must capture this knowledge, and the matching algorithm must support, a more sophisticated concept of context and the sequence of RSM

operators before this process can be completely automated.

Constructive induction is a knowledge intensive learning process. Further research should provide even more advanced capabilities for introducing and employing domain knowledge to guide constructive induction (Ragavan and Rendell, 1991). For example, there should be a facility for a user to indicate different preferences for various types of constructive induction operators. Also, it should be easy to the user to give advice as to the use of some new type of operators.

This raises a general issue of how to include within a constructive induction system sophisticated knowledge representation capabilities. Consequently, there is a need for developing a general method for what type of knowledge should be represented that might be useful for creating a more adequate knowledge representation, and how it should be used.

AQ17-MCI uses rule-based knowledge representation system. An interesting issue is to investigate how various ideas and operators implemented in AQ17-MCI could be employed in learning systems using different knowledge representation, e.g., decision trees, semantic network, neural nets, etc. To employ any type of modification operator withing another representation language will need to deal with the problems already addressed here, such as detection and reduction of the overprecision of data, noise in the training data, or low quality data (e.g., many irrelevant attributes). It is believed that the same cues used to select transformations relevant to a rule-based representation will be useful for other representation languages.

The above raises a general issue of developing a constructive induction learning system that employs multi-type representation language. This would allow the system to represent different types of knowledge in the form that is most suitable to them.

Acknowledgments

The authors wish to thank Mike Hieb and Ken Kaufman for their helpful comments and critical review of this paper.

This research was conducted in the Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the National Science Foundation under grant No. IRI-9020266, in part by the Advanced Research Projects Agency under the grant No. N00014-91-J-1854, administered by the Office of Naval Research, and the grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under grant No. N00014-91-J-1351.

References

- Aha, D.W., "Generalizing from Case Studies: A Case Study," *Ninth International Workshop on Machine Learning*, pp. 1-10, Edinburgh, Scotland, 1992.
- Arciszewski, T., Bloedorn, E., Michalski, R.S., Mustafa M. and Wnek J., "Constructive Induction in Structural Design," *ASCE Journal of Computing in Civil Engineering (submitted)*, Vol. pp. 1992.
- Bergadano, F., Matwin, S., Michalski, R.S., Zhang, J., "A General Criterion for Measuring Quality of Concept Descriptions," Center for AI, George Mason University, MLI 88-31, 1988.
- Bloedorn, E., and Michalski, R.S., "Constructive Induction from Data in AQ17-DCI: Further Experiments," Reports of the Machine Learning and Inference Laboratory, MLI 91-12, Center for AI, GMU, 1991.
- Holder, L., "Selection of Learning Methods Using an Adaptive Model of Knowledge Utility," *Proceedings of the First International Workshop on Multistrategy Learning*, pp. 247-254, Harper's Ferry, WV, 1991.
- Holte, R.C., Acker, L.E., and Porter, B.W., "Concept Learning and the Problem of Small Disjuncts," *Proceedings of IJCAI-89*, pp. 813-818, Detroit, MI, 1989.
- Imam, I.F., and Michalski, R.S., "Learning Decision Trees from Rules: A Comparative Study", *IIIS Journal of Intelligent Information Systems*, L., Kerschberg, Z., Ras, and M., Zemankova, (eds.), Kluwer Academic, 1993.
- Kaufman, K., Michalski, R.S., and Kerschberg, L., "Knowledge Extraction from Databases: Design Principles of the INLEN System," *Sixth International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC, 1991.
- Kerber, R., "ChiMerge: Discretization of Numeric Attributes," *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 123-128, San Jose, CA, 1992.
- Matheus, C.J., "The Need for Constructive Induction," *Proceedings of the Eighth International Workshop (ML91)*, pp. 173-177, Evanston, IL, 1991.
- Matheus, C.J., and Rendell, L., "Constructive Induction on Decision Trees," *Proceedings of IJCAI-89*, pp. 645-650, Detroit, MI, 1989.
- Mehra, P., Rendell, L., and Benjamin, W., "Principled Constructive Induction," *Proceedings of IJCAI-89*, pp. 651-656, Detroit, MI, 1989.
- R. S. Michalski, "Pattern Recognition as Knowledge-Guided Computer Induction," Department of Computer Science Reports, No. 927, University of Illinois, Urbana, June 1978.
- Michalski, R.S., "A Theory and Methodology of Inductive Learning," *Machine Learning: An Artificial Intelligence Approach*, Vol. I, R.S. Michalski, J.G. Carbonell and T.M. Mitchell

(Eds.), Palo Alto, CA: Morgan Kaufmann, 1983.

Michalski, R. S., "Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-Tiered Representation" *Machine Learning: An Artificial Intelligence Approach* vol. 3, Y. Kodratoff and R.S. Michalski (eds.), pp. 63-102. San Mateo: Morgan Kaufmann, 1983.

Michalski, R.S., "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning," in *Machine Learning: A Multistrategy Approach*, Vol. IV, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA, 1993.

Mitchell, T.M., Utgoff, P.E., and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics", *Machine Learning: An Artificial Intelligence Approach, Vol. I*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), Palo Alto, CA: Morgan Kaufmann, 1983.

Pachowicz, P.W., Bala, J., and Zhang, J., "Iterative Rule Simplification for Noise-Tolerant Inductive Learning," *Proceedings of the Fourth International Conference on Tools for Artificial Intelligence*, pp. 452-453, Arlington, VA, 1992.

Pagallo, G., and Haussler, D., "Boolean Feature Discovery in Empirical Learning," *Machine Learning*, 1990.

Quinlan, J.R., "Learning Efficient Classification Procedures and their Application to Chess End Games," *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski Carbonell, J.G., and Mitchell, T. (Eds.), Palo Alto, CA: Morgan Kaufmann, 1983.

Ragavan, H. and Rendell, L., "Relations, Knowledge and Empirical Learning," *Proceedings of the Eighth International Workshop on Machine Learning (ML91)*, pp. 188-192, Evanston, IL, 1991.

Reinke, R.E., *Knowledge Acquisition and Refinement Tools for the ADVISE Meta-Expert System*, University of Illinois at Urbana-Champaign, Master's Thesis, 1984.

Rendell, L. and Seshu, R., "Learning Hard Concepts Through Constructive Induction: Framework and Rationale," *Computer Intelligence*, Vol. 6 pp. 247-270. 1990.

Schlimmer, J.C., "Learning and Representation Change," *Proceedings of AAAI-87*, pp. 511-515, Seattle, WA, 1987.

Spears, W., and Gordon, D., "Adaptive Strategy Selection for Concept Learning," *Proceedings of the First International Workshop on Multistrategy Learning*, pp. 231-246, Harper's Ferry, WV, 1991.

Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzerowski, S., Fahlman, S.E., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J., Zhang, J., "The MONK'S Problems: A Performance Comparison of Different Learning Algorithms," (*revised version*), Carnegie Mellon University, Pittsburgh, PA, CMU-CS-91-197, 1991.

Watanabe, L. and Elio, R., "Guiding Constructive Induction for Incremental Learning from Examples," *Proceedings of IJCAI-87*, pp. 293-296, Milan, Italy, 1987.

Wnek, J. and Michalski, R.S., "Hypothesis-Driven Constructive Induction in AQ17: A Method and Experiments," *Proceedings of the IJCAI-91 Workshop on Evaluating and Changing Representation in Machine Learning*, Sydney, Australia, August 1991.

Wnek, J., "Hypothesis-driven Constructive Induction," *Ph.D. dissertation*, School of Information Technology, *Reports of Machine Learning and Inference Laboratory*, MLI 93-2, Center for Artificial Intelligence, George Mason University, March 1993.

Wnek, J. and Michalski, R.S., "Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments," *Machine Learning*, Special Issue on Representation, to appear 1993.

IV. Multiple Computational Strategies

Extracting Symbolic Rules from Artificial Neural Networks

Mark W. Craven and Jude W. Shavlik

Computer Sciences Department

University of Wisconsin

1210 West Dayton St.

Madison, WI 53706

email: {craven, shavlik}@cs.wisc.edu

Abstract

A distinct advantage of symbolic learning algorithms over artificial neural networks is that typically the concept representations they form are more easily understood by humans. A multistrategy approach to understanding the representations formed by neural networks is to extract symbolic rules from trained networks. In this paper we describe and investigate an approach for extracting rules from networks that uses the NOFM extraction algorithm and the network training method of soft weight-sharing. The NOFM algorithm had previously been successfully applied only to *knowledge-based* neural networks. Our experiments demonstrate that our extracted rules generalize better than rules learned using the C4.5 algorithm. In addition to being accurate, our extracted rules are also reasonably comprehensible.

Keywords: artificial neural networks, rule extraction, empirical comparison

1 Introduction

Artificial neural networks (ANNs) have been successfully applied to real-world problems as varied as steering a motor vehicle (Pomerleau, 1991) and learning to pronounce English text (Sejnowski & Rosenberg, 1987). In

addition to these practical successes, several empirical studies have concluded that neural networks provide performance comparable to, and in some cases, better than common symbolic learning algorithms (Fisher & McKusick, 1989; Mooney et al., 1989; Weiss & Kapouleas, 1989). A distinct advantage of symbolic learning algorithms, however, is that the concept representations they form are usually more easily understood by humans than the representations formed by neural networks. In this paper we describe and investigate a multistrategy approach to inductive learning that involves extracting symbolic rules from trained neural networks. Our approach uses the NOFM algorithm (Towell & Shavlik, 1991) to extract rules from networks that have been trained using Nowlan and Hinton's method (1992) of *soft weight-sharing*. We present experiments that demonstrate that, for two difficult learning tasks, our method learns rules that are more accurate than rules induced by Quinlan's C4.5 algorithm (1993). Furthermore, the rules that are extracted from our trained networks are comparable to rules induced by C4.5 in terms of complexity and understandability.

Towell and Shavlik (1991) demonstrated that concise and accurate symbolic rules

can be extracted from *knowledge-based* neural networks. In a knowledge-based network, the topology and initial weights of the network are specified by a domain theory consisting of symbolic inference rules. Since these networks initially encode symbolic rules, training is more a process of rule refinement than of *tabula rasa* learning. This paper describes work that involves using Towell and Shavlik's NOFM algorithm to extract rules from ANNs which have *not* been initialized by a domain theory. Because the NOFM algorithm assumes that the weights in a trained network are clustered, we modify the training process to encourage such a network state after training. Previously, Towell (1991) reported that the NOFM algorithm failed to extract accurate rules from conventional networks.

We use two problem domains to investigate the effectiveness of our approach. The first domain involves recognizing *promoters* in DNA (Towell et al., 1990). Promoters are short nucleotide sequences that occurs before genes and serve as binding sites for the protein *RNA polymerase* during gene transcription. Identifying promoters is an important step in locating genes in DNA sequences. The second problem domain that we investigate is a simplified version of the NETTALK task of mapping English text to its pronunciation (Sejnowski & Rosenberg, 1987). Our scaled-down version of this domain involves learning only the stresses (but not the phonemes) from a corpus of the 1000 most common English words.

The organization of this paper is as follows: the next section discusses the problem of extracting rules from neural networks and describes the NOFM algorithm that is employed in our approach. Section 3 describes soft weight-sharing and how we use for the task of rule extraction. Section 4 describes two problem domains that we use to investigate the effectiveness of our approach, and

section 5 presents experimental results for these domains. The final section provides conclusions and a discussion of future work.

2 Extracting Rules From Neural Networks

An important criterion by which a machine learning algorithm should be judged is the comprehensibility of the representations formed by the algorithm. That is, does the algorithm encode the information it learns in such a way that it may be inspected and understood by humans? There are at least five reasons why this is an important criterion.

- *Validation.* If the designers and end-users of a learning system are to be confident in the performance of the system, then they must understand how it arrives at its decisions.
- *Discovery.* Learning algorithms may discover salient features in the input data whose importance was not previously recognized. If the representations formed by the algorithm are comprehensible, then these discoveries can be made accessible to human review.
- *Explanation.* If the representations are understandable, then an explanation of the classification made on a particular case can be garnered.
- *Improving generalization.* The feature representation used for an inductive learning task can have a significant impact on generalization performance. Understanding learned concept representations may facilitate the design of a better feature representation for a given problem.
- *Refinement.* Some researchers use inductive learning systems to refine an

approximately-correct domain theory (Ourston & Mooney, 1990; Towell et al., 1990). When a learning system is used in this way, it is important to understand the changes to the knowledge base that have been imparted during the training process.

2.1 Rule Extraction Methods

A significant limitation of artificial neural networks is that the concepts they learn are virtually impenetrable to human understanding because concepts are represented by a large number of real-valued parameters: the weights and biases of the network. One approach to understanding the representations formed by a neural network is to use scientific visualization techniques (Wejchert & Tesauero, 1990). A second approach, which is applicable to small feature spaces, involves a combination of visualization and rule extraction (Wnek & Michalski, 1991). The approach on which we focus in this paper is the extraction of symbolic rules from networks of arbitrary size (Fu, 1991; McMillan et al., 1991; Saito & Nakano, 1988).

The underlying premise of these rule-extraction methods is that each hidden and output unit in the network can be thought of as implementing a symbolic rule. The concept associated with each unit is the consequent of the rule, and certain subsets of the units that feed into this unit represent the antecedents of the rule. As shown in Figure 1, the process of rule extraction involves finding the *sufficient* conditions for each consequent. In order to find such sets of sufficient conditions, rule-extraction methods assume that, after training, hidden and output units tend to be either maximally active (i.e., have activation near one), or inactive (i.e., have activation near zero). Given this assumption, a rule-extraction algorithm can

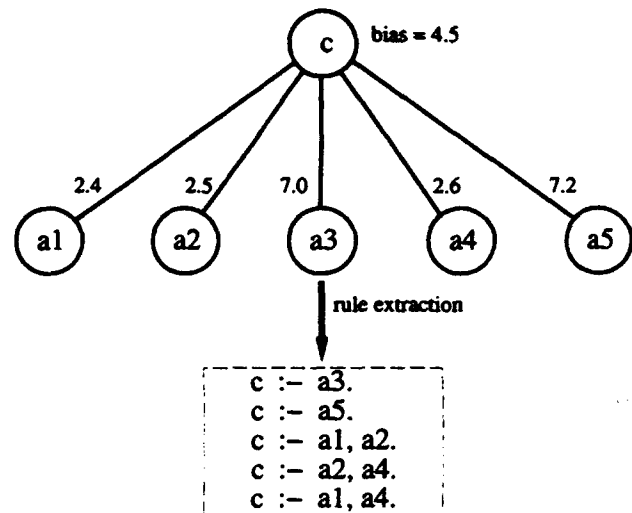


Figure 1: Extracting rules from a unit in a neural network. The extracted rules show which combinations of antecedent units must be activated in order for the consequent unit's bias to be exceeded.

search for minimal sets of antecedent units that, when maximally active, cause the consequent unit to become maximally active. The process of searching for rules is problematic because of the combinatorics involved. The complexity of this search, in the worst case, is $O(2^n)$ where n is the number of connections impinging on the consequent unit. Moreover, these algorithms tend to extract a large number of rules, even for networks of moderate complexity.

2.2 The NofM Algorithm

Towell and Shavlik previously described an algorithm, called NOFM, that avoids the combinatoric and rule-set size problems of other rule-extraction algorithms by clustering weights into equivalence classes. They have demonstrated that their NOFM algorithm is able to extract accurate and concise rules from trained knowledge-based neural networks; that is, networks for which the topology and initial weights have been specified by an approximately-correct domain theory. The algorithm is called NOFM be-

cause it explicitly searches for rules of the form:

If (N of the M antecedents are true) then ...

The NOFM algorithm comprises six steps:

1. **Clustering.** The weights impinging on each hidden and output unit of the trained network are grouped into clusters. Initially, each weight is treated as a cluster. The two nearest clusters are successively merged until no pair of clusters is closer than a preselected distance. Additionally, weights with small magnitudes are pruned from the network at this step.
2. **Averaging.** The magnitude of each weight is set to the average value of the weights in its cluster.
3. **Eliminating.** Weight clusters that are not needed in order to correctly activate a unit are eliminated. Two elimination procedures are applied: one algorithmic and one heuristic. The algorithmic elimination procedure identifies clusters that *cannot* have an effect on whether or not a unit's bias is exceeded. The heuristic elimination step eliminates clusters that do not have such an effect for any of the training examples.
4. **Optimizing.** The unit biases are retrained to adapt the network to the changes that have been imparted by the previous steps.
5. **Extracting.** Each hidden and output unit is translated into a rule with weighted antecedents such that the consequent is true if the sum of the weighted antecedents exceeds the bias.
6. **Simplifying.** Weights and thresholds are eliminated and rules are expressed in the NOFM format.

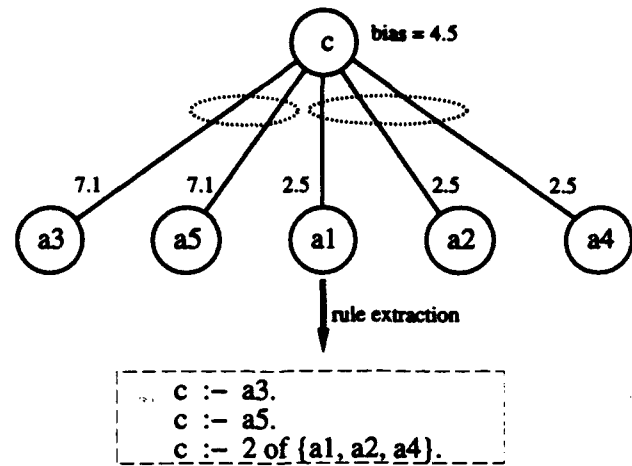


Figure 2: Extracting rules using the NOFM method. The dotted ovals illustrate how the weights have been grouped into clusters. Each weight has been set to the average value of its cluster.

Figure 2 illustrates the application of the NOFM to the unit shown in Figure 1. The weights have been grouped into two clusters, and each weight has been set to the average value of its cluster. One of the extracted rules is expressed in the NOFM format; the other two rules are trivial NOFM cases (1 of 1). The *eliminating* and *optimizing* steps are not depicted in this example.

3 Extending NofM With Soft Weight-Sharing

An underlying assumption of the NOFM method is that the distribution of weights in the network will be conducive to forming a small number of clusters for each hidden and output unit. For knowledge-based neural networks, this is a reasonable assumption since the weights are clustered at least before training. For example, using the KBANN algorithm for mapping symbolic rules into a knowledge-based network (Towell et al., 1990), the weights that are specified by the domain theory have values of approximately 4 and -4, whereas the rest of the weights

have values near 0. Experimental evidence indicates that the weights tend to be fairly well clustered after training as well (Towell, 1991).

The applicability of the NOFM method might seem to be limited to knowledge-based networks since in conventional neural networks there is usually not a bias that leads weight values to be clustered after training. In fact, Towell (1991) reported that NOFM did not extract small sets of accurate rules from conventional networks. However, the approach that we explore in this paper does not rely on the network weights being initially clustered, but instead encourages clustering during network training. We use a method developed by Nowlan and Hinton (1992), termed *soft weight-sharing*, that encourages weights to form clusters during the training process. Although their method was motivated by the desire for better generalization, we explore it here as a means for facilitating rule extraction.

Soft weight-sharing uses a cost function that penalizes network complexity so that during training, the network tries to find an optimal tradeoff between data-misfit (i.e., the error rate on the training examples) and complexity. The complexity term in soft weight-sharing models the distribution of weights in the network as a mixture of multiple Gaussians. A set of weights is considered to be simple if the weights have high probability densities under the mixture model. Specifically, the cost function in soft weight sharing is the following:

$$C = \lambda E - \sum_i \log \left[\sum_j \pi_j p_j(w_i) \right]$$

where E is the data-misfit term, λ is a parameter used to balance the tradeoff between data misfit and complexity, w_i is a weight in the network, $p_j(w_i)$ is the density value of w_i under the j th Gaussian, and π_j is the mix-

ing proportion of the j th Gaussian. A mixing proportion is a weight that determines the influence of a particular Gaussian. The mixing proportions are constrained to sum to 1.

The partial derivative of the cost function with respect to each weight is the sum of the usual error derivative plus a term due to the complexity cost of the weight:

$$\frac{\partial C}{\partial w_i} = \lambda \frac{\partial E}{\partial w_i} - \sum_j r_j(w_i) \frac{(\mu_j - w_i)}{\sigma_j^2}$$

Here μ_j and σ_j^2 are the mean and variance, respectively, of the j th Gaussian, and $r_j(w_i)$ is the conditional probability that w_i is being modelled by the j th Gaussian:

$$r_j(w_i) = \frac{\pi_j p_j(w_i)}{\sum_k \pi_k p_k(w_i)}$$

Thus, the effect of each Gaussian is to pull each weight toward the mean of the Gaussian with a force proportional to the density of the Gaussian at the value of the weight. When weights are pulled tightly around the means of the Gaussians, the network is similar to one that has fewer free parameters than connections (ordinary weight sharing). The parameters of each Gaussian – the mean μ_j , standard deviation σ_j , and mixing proportion π_j – are learned simultaneously with the weights during training.

Our approach to rule extraction involves training networks using a variant of soft weight-sharing and then applying the NOFM algorithm to the trained networks. Although the NOFM method was designed for knowledge-based neural networks, we hypothesized that it could be successfully applied to conventional networks, provided that the weights of the networks were grouped into clusters during training.

Whereas the NOFM algorithm works best when the weights impinging on *each unit* form clusters, soft weight-sharing tends to

globally cluster network weights. Our implementation of soft weight-sharing, however, assigns a local set of Gaussians to each unit. The complexity cost of a given weight is calculated with respect to only the Gaussians associated with the unit to which the weight connects.

4 Data Sets

Our experiments address the hypothesis that soft weight-sharing is able to cluster the network weights during training such that NOFM is able to extract a small set of accurate rules. In order to evaluate the effectiveness of our approach, we use two problem domains to compare the accuracy and succinctness of our extracted rules against rules induced by the C4.5 algorithm (Quinlan, 1993). Both problem domains involve predicting a class given a fixed-length "window" onto a string of interest. In the case of the *promoter* domain, the string is a DNA sequence, and in the NETTALK domain, the string is an English word (or part of one).

The *promoter* data set comprises 468 examples¹, half of which are positive examples (i.e., promoters). Each example has 57 features which represent the DNA sequence. A single strand of DNA is a linear chain composed from the four nucleotides represented by the letters A, C, G, T. Thus all of the features for this problem are nominal features that can take on the values A, C, G, T, or *unknown*. Each example is a member of one of two classes, *promoter* or *non-promoter*. Recall that a promoter occurs before a gene on a DNA strand. The positive examples for this data set are aligned such that the gene following each promoter begins in the sev-

enth position from the right end of the window. Thus the leftmost 50 window positions are labelled -50 to -1, and the rightmost seven are labelled 1 to 7.

For the neural networks, a local representation is used for the *promoter* features. Thus, for each feature there are four input units – one corresponding to each of the nucleotides. When the value of a feature is known, the input unit corresponding to the value is given an activation of 1, and the other three units for the feature are given activations of 0. When a feature value is unknown, all four input units are given activations of 0.25.

Our simplified NETTALK data set consists of 5438 examples taken from the 1000 most common English words. Each example has seven features which represent the letters in the input window. Each feature can take on one of 27 values. There is a value corresponding to each letter of the alphabet, and a value to represent the absence of a letter. Since each example is formed from only a single word, when the window overhangs a word, the overhanging window positions are set to the "space" value. In the original NETTALK domain, the task involved predicting both a phoneme and a stress for each window position. In our experiments we have simplified the problem so that the classifiers are trained only to predict a stress (out of five disjoint classes).

5 Experimental Results

In this section we evaluate our approach to rule extraction by comparing the accuracy and comprehensibility of rules extracted from neural networks and rules learned using the C4.5 algorithm. The comprehensibility of a set of rules is a difficult concept to measure. We simply measure the syntactic complexity of the rule sets and use this as

¹Note that this data set is larger than the one that was used in (Towell et al., 1990) and is available by anonymous ftp from the UC-Irvine Repository of Machine Learning Databases and Domain Theories (ftp.ics.uci.edu). The *promoter* set in the Irvine database contains only 106 examples.

a proxy for complexity. Specifically, we consider the number of rules and antecedents to be measures of syntactic complexity.

5.1 The Promoter Data Set

For the *promoter* problem, we use a ten-fold cross-validation methodology² to assess the ability of our approach to extract accurate, comprehensible rules from trained networks. Our reported results represent averaged values for the ten runs.

The neural networks used for the *promoter* domain have fully-connected hidden units in a single layer. The number of hidden units used in each network is determined by cross-validation within the training set. That is, for each training set, networks with 20, 25, 10, and no hidden units are trained, and cross-validation is used to pick the network that is to be trained on all of the data in the training set. After the number of hidden units is selected for each network, a similar cross-validation procedure is used to determine the λ parameter for soft weight-sharing. We use a conjugate-gradient learning algorithm to train the weights and the Gaussian parameters of the networks.

Decision trees are induced, and rules extracted from them, using Quinlan's C4.5 algorithm (1993). Cross-validation within each training set is used to determine the confidence levels for both tree pruning and rule pruning. The confidence level selected for tree pruning does not affect the rule extraction results since the C4.5 rule-induction program operates on unpruned trees and performs its own pruning independently of the tree-induction program. However, we select a confidence level for tree pruning only so that we obtain an accurate

Table 1: Generalization on the *promoter* data.

approach		% test set error
C4.5	decision trees	16.9
	rules	13.5
ANNs	networks	7.9
	rules	11.1

Table 2: Rule-set sizes for the *promoter* data.

approach	# rules	# antecedents
C4.5	23.2	47.3
ANNs	8.2	119.6

estimate of decision tree generalization for this task. For each training set, we test confidence levels ranging from 5% to 95% and separately select tree-pruning and rule-pruning levels.

Table 1 shows the test set error rates on the *promoter* data set for the decision trees, rules extracted from the trees, neural networks, and rules extracted from the networks. As can be seen in the table, neural networks perform significantly better on this task than decision trees or the rules extracted from them. Additionally, the performance of the symbolic rules extracted from the neural networks is fairly close to the performance of the networks themselves, and better than the rules extracted from the decision trees. The difference in error rates between the rules extracted from networks and the C4.5 rules is significant at the 0.05 level using a paired, 1-tailed *t*-test.

Table 2 shows the average number of rules and antecedents for the rules extracted from our networks and the rules induced by C4.5. The rule sets extracted from the decision trees contain more rules but fewer antecedents than the sets extracted from networks. The additional complexity of the rules extracted from networks, however, re-

²In ten-fold cross-validation, the available data is partitioned into ten sets. Classifiers are trained using examples from nine of the sets and tested on examples from the tenth set. This procedure is repeated ten times so that each set is used as the testing set once.

sults in a significant gain in accuracy. Moreover, the rules extracted from networks have only 14.6 antecedents per rule, so we believe that their complexity is within the bounds of what biologists can readily understand.

Table 3 shows a set of rules extracted from one of the *promoter* networks. In addition to the NOFM-style rules, this rule set has been expressed using a predicate we call *more_than*. The *more_than* predicate has the following form:

$N \text{ more_than}(\text{Pos_Set}, \text{Neg_Set})$

where N is an integer, and *Pos_Set* and *Neg_Set* are sets of positive and negated antecedents respectively. The predicate returns true if the number of true antecedents in *Pos_Set* minus the number of true antecedents in *Neg_Set* is greater than N . This predicate provides a succinct way of expressing rules that have many negated antecedents. Without such a predicate, negated antecedents tend to result in a large number of mostly-redundant rules. Since the knowledge-based networks to which the NOFM algorithm was previously applied had very few negated antecedents, this predicate was not previously necessary.

The rule set shown in Table 3 exhibits several interesting characteristics. First, the rules abstract away a significant amount of the complexity of the network from which they are extracted. There are only ten rules and a total of 106 antecedents. Six of the hidden units and more than 2400 of the weights that were present in the neural network are not represented in the rules. A second observation is that the rules focus on what are known by biologists to be the most significant regions of the DNA sequence. In particular, a domain theory developed by Michiel Noordewier (Towell et al., 1990) identifies the -14 to -7 and the -37 to -31 regions as containing the most important fea-

Table 3: Rules extracted from a *promoter* network. The predicate *more_than* returns true if the number of true antecedents in the first set minus the number of true antecedents in the second set is greater than the supplied threshold. The notation @-36 indicates the starting position of a given sequence; in this case the position is 36 nucleotides before the start of a putative gene. Dashes represent placeholders, so {@-36 '---AG---A'} has only three antecedents. The letter S is an ambiguity code that biologists use to represent (C \vee G).

```

promoter :-
    hidden_2,
    not (hidden_1),
    not (hidden_4).

promoter :-
    hidden_3,
    not (hidden_1),
    not (hidden_4).

promoter :-
    hidden_2,
    hidden_3,
    not 2 of {hidden_1, hidden_4}.

hidden_1 :-
    5 more_than( {@-40 'C---C-G-C-G',
                  @-13 '-SG---', @-1 'G'},
                {@-40 'A---T---A---',
                  @-13 '-T---T'} ).

hidden_1 :-
    not ({@-40 '-----T-----'}),
    3 more_than( {@-40 'C---C-G-C-G',
                  @-13 '-SG---', @-1 'G'},
                {@-40 'A---T---A---',
                  @-13 '-T---T'} ).

hidden_2 :-
    5 more_than( {@-40 'A---T-GA-A',
                  @-13 '-T-'},
                {@-40 'C---C-G---',
                  @-13 '-SG'} ).

hidden_2 :-
    {@-40 '-----T-----'},
    3 more_than( {@-40 'A---T-GA-A',
                  @-13 '-T-'},
                {@-40 'C---C-G---',
                  @-13 '-SG'} ).

hidden_3 :-
    4 more_than( {@-40 'A---T-GA-AT',
                  @-13 '-T-A-T'},
                {@-40 '-----C-G-C-',
                  @-20 'G--G',
                  @-13 'GSGG'} ).

hidden_3 :-
    {@-40 '-----T-----'},
    2 more_than( {@-40 'A---T-GA-AT',
                  @-13 '-T-A-T'},
                {@-40 '-----C-G-C-',
                  @-20 'G--G',
                  @-13 'GSGG'} ).

hidden_4 :-
    4 of {@-44 'G', @-40 '---AC-G-C',
          @-24 'A--G', @-13 '-SG'}.

```

tures of a promoter. These are termed the *contact* regions. The rules extracted from all of the hidden units specify antecedents primarily in these areas.

5.2 The Nettalk Data Set

For the NETTALK domain, classifiers are trained on half of the example set and tested on the other half. Ten runs of this procedure are performed, and the reported results represent averaged values. As with the *promoter* domain, cross-validation is used to determine the λ parameter and the number of hidden units for the networks, and the confidence levels for pruning C4.5 trees and rules.

Since the NETTALK domain involves five classes, the rules extracted from trained networks are not necessarily mutually exclusive and exhaustive. In other words, a given input sequence may satisfy more than one of the class rules, or alternatively, it may satisfy none of the class rules. The C4.5 rule-extraction method also faces this complication when it prunes antecedents and rules from its rule set. C4.5 handles this problem in two ways: (1) rules are ordered by class, and the first rule to match a given instance determines the predicted class; (2) a default rule is used to classify instances that do not satisfy any of the other rules. We employ a similar policy in classifying instances using our network-extracted rules. The rules are ordered according to the *a priori* probability of each class, and the default rule predicts the most probable class.

Table 4 shows the test set error rates on the NETTALK data set for the decision trees, rules extracted from the trees, neural networks, and rules extracted from the networks. The results in this table indicate that the neural networks and the rules extracted from them outperform C4.5 decision trees and rules. The difference in error rates between the rules extracted from networks and

Table 4: Generalization on the NETTALK data.

approach		% test set error
C4.5	decision trees	19.1
	rules	20.1
ANNs	networks	13.0
	rules	17.0

Table 5: Rule-set sizes for the NETTALK data.

approach	# rules	# antecedents
C4.5	233.5	466.5
ANNs	17.5	661.9

the decision trees is significant at the 0.005 level using a paired, 1-tailed *t*-test.

Table 5 shows the average number of rules and antecedents in the extracted rule sets. The rule sets extracted from the neural networks contain far fewer rules than the rules generated from the decision trees, although the network rules have far more antecedents.

6 Conclusions

We have demonstrated that small sets of accurate, reasonably concise symbolic rules can be extracted from ordinary artificial neural networks. Our approach to this problem involves exploiting the effectiveness of the NOFM algorithm by encouraging weight clustering during training. For two difficult problem domains, recognizing promoters in DNA, and mapping English text to stress patterns, our approach was able to induce rules that resulted in better generalization than rules learned using the C4.5 symbolic learning algorithm.

There are a number of issues regarding our approach that we plan to pursue in further research. One such issue is adapting the approach so that it can extract concise rule sets from networks that have learned

distributed representations. In a distributed representation, each concept at the hidden-unit level may be encoded by the activations of many hidden units, and each unit may play a part in representing many different concepts. Distributed representations tend to result in rule sets that are verbose and difficult to understand. The NOFM algorithm makes the assumption that each hidden unit corresponds to a meaningful concept (which is an appropriate assumption for knowledge-based networks), and thus it searches for rules by considering each hidden unit independently. Our proposed approach involves partitioning the space of hidden unit activations and then searching for rules that explain particular regions of this space.

A second area that we plan to investigate in future research is to employ a weight-pruning method, such as *Optimal Brain Damage* (LeCun et al., 1990), during learning. The effectiveness of the NOFM algorithm is partly due to the weight pruning that it performs during its clustering step. The expected advantage of pruning weights during the learning process is that the remaining weights are able to adapt to the changes imparted by the pruning operation. An additional advantage of the Optimal Brain Damage technique is that it does not base its pruning decisions on weight magnitudes, but instead bases them on the second partial derivative of each weight with respect to the cost function. The sensitivity of the cost function to a given weight is better estimated by a second derivative than by a magnitude.

Another area for future research is to investigate the range of domains to which our approach can be successfully applied.

Extracting accurate, comprehensible rules from neural networks is an important problem in machine learning. We have described an approach that employs the NOFM algorithm and soft weight-sharing, and demon-

strated that it is able to extract accurate, comprehensible rules from networks trained on a difficult real-world problem. These promising results indicate that the problem of understanding representations learned by artificial neural networks may be tractable.

Acknowledgements

This work was partially supported by Department of Energy Grant DE-FG02-91ER61129 and National Science Foundation Grant IRI-9002413.

References

- Fisher, D. H. & McKusick, K. B., An empirical comparison of ID3 and back-propagation, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 788-793), Detroit, MI, 1989.
- Fu, L. M., Rule learning by searching on adapted nets, *Proceedings of the Ninth National Conference on Artificial Intelligence*, (pp. 590-595), Anaheim, CA, 1991.
- LeCun, Y., Denker, J. S., & Solla, S. A., Optimal brain damage, in *Advances in Neural Information Processing Systems (volume 2)*, (pp. 598-605), Morgan Kaufmann, 1990.
- McMillan, C., Mozer, M., & Smolensky, P., The connectionist scientist game: Rule extraction and refinement in a neural network, *Proceedings of the Thirteenth Conference of the Cognitive Science Society*, Chicago, IL, 1991.
- Mooney, R., Shavlik, J., Towell, G., & Gove, A., An experimental comparison of symbolic and connectionist learning algorithms, *Proceedings of the Eleventh International Joint Conference on Artificial In-*

telligence, (pp. 775-780), Detroit, MI, 1989, (Longer version appears in *Machine Learning*, 6(2)).

Nowlan, S. J. & Hinton, G. E., Simplifying neural networks by soft weight-sharing, *Neural Computation*, 4:473-493, 1992.

Ourstun, D. & Mooney, R. J., Changing the rules: A comprehensive approach to theory refinement, *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 815-820), Boston, MA, 1990.

Pomerleau, D. A., Efficient training of artificial neural networks for autonomous navigation, *Neural Computation*, 3:88-97, 1991.

Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

Saito, K. & Nakano, R., Medical diagnostic expert system based on PDP model, *Proceedings of the IEEE International Conference on Neural Networks (volume 1)*, (pp. 255-262), San Diego, CA, 1988.

Sejnowski, T. & Rosenberg, C., Parallel networks that learn to pronounce English text, *Complex Systems*, 1:145-168, 1987.

Towell, G. G., *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction*, PhD thesis, University of Wisconsin - Madison, 1991.

Towell, G. G. & Shavlik, J. W., Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules, in *Advances in Neural Information Processing Systems (volume 4)*, Morgan Kaufmann, 1991.

Towell, G. G., Shavlik, J. W., & Noordewier, M. O., Refinement of approximately correct domain theories by

knowledge-based neural networks, *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 861-866), Boston, MA, 1990.

Weiss, S. M. & Kapouleas, I., An empirical comparison of pattern recognition, neural nets, and machine learning classification methods, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 688-693), Detroit, MI, 1989.

Wejchert, J. & Tesauro, G., Neural network visualization, in *Advances in Neural Information Processing Systems (volume 2)*, (pp. 465-472), Morgan Kaufmann, 1990.

Wnek, J. & Michalski, R. S., An experimental comparison of symbolic and sub-symbolic learning paradigms: Phase I - learning logic-style concepts, *Proceedings of the First International Workshop on Multistrategy Learning*, (pp. 324-339), Harpers Ferry, WV, 1991.

A Multistrategy Learning Scheme for Assimilating Advice in Embedded Agents

Diana F. Gordon
Naval Research Laboratory, Code 5514
Washington, D.C. 20375
gordon@aic.nrl.navy.mil

Devika Subramanian
Cornell University
Ithaca, N.Y. 14853
devika@cs.cornell.edu

Abstract

The problem of designing and refining task-level strategies in an embedded multiagent setting is an important unsolved question. To address this problem, we have developed a multistrategy system that combines two learning methods: operationalization of high-level advice provided by a human and incremental refinement by a genetic algorithm. The first method generates seed rules for finer-grained refinements by the genetic algorithm. Our multistrategy learning system is evaluated on two complex simulated domains as well as with a Nomad 200 robot.

Key words: advice, operationalize, genetic algorithms

1 Introduction

The problem of designing and refining task-level strategies in an embedded multi-agent setting is an important unsolved question. To address this problem, we have developed a multistrategy learning system that combines two learning methods: operationalization of high-level advice provided by a human, and incremental refinement by a genetic algorithm (GA). We define advice as a

recommendation to achieve a goal under certain conditions. Advice is considered to be operationalized when it is translated into stimulus-response rules in a language directly usable by the agent. Operationalization generates seed rules for finer-grained refinements by a GA.

The long term goal of the work proposed here is to develop task-directed agents capable of acting, planning, and learning in worlds about which they have incomplete information. These agents refine factual knowledge of the world they inhabit, as well as strategic knowledge for achieving their tasks, by interacting with the world. *Agent knowledge acquisition* is very difficult for the same reasons that knowledge acquisition for expert systems is. It is preferable to assimilate high level knowledge because the process of entering low level domain-specific knowledge for an agent is a costly, tedious, and error-prone process. The additional challenge for agent knowledge acquisition comes from the fact that the agent must dynamically update its knowledge through interactions with its environment.

There are two basic approaches to constructing agents for dynamic environments. The first decomposes the design into stages: a

parametric design followed by refinement of the parameter values using feedback from the world in the context of the task. Several refinement strategies have been studied in the literature: GAs (Odetayo and McGregor, 1989), neural-net learning (Clouse and Utgoff, 1992), statistical learning (Maes and Brooks, 1990), and reinforcement learning (Mahadevan and Connell, 1991). The second, more ambitious, approach (Grefenstette *et al.*, 1990; Tesauro, 1992) is to acquire the agent knowledge directly from example interactions with the environment. The success of this approach is tied to the efficacy of the credit assignment procedures, and whether or not it is possible to obtain good training runs with a knowledge-impooverished agent.

We have adopted the first approach. The direction we pursue is to compile an initial parametric agent using high-level strategic knowledge (e.g., advice) input by the user, as well as a body of general (not domain-specific) spatial knowledge in the form of a *Spatial Knowledge Base* (SKB). The SKB contains qualitative rules about movement in space. Example rules in our SKB are "If something is on my side, and I turn to the other side, I will not be facing it" and "If I move toward something it will get closer". This SKB is portable because it is applicable to a variety of domains where qualitative spatial knowledge is important. A similar qualitative knowledge base was constructed by (Mitchell, 1987) for the task of pushing objects in a plane. Since the knowledge provided to our agent will often be imperfect (incomplete and incorrect), this knowledge is refined by a GA.

First we describe our deductive operationalization process and the nature of the parameterization adopted for our agent. Then we

describe the inductive (GA) refinement stage and compare our multistrategy approach with one that is purely inductive. Before we present the details of the method, we characterize the class of environments and tasks for which we have found this decomposition (of an agent design into an initial parametric stage and subsequent refinement stage) to be effective.

- **Environment characteristics:** Complete models of the dynamics of the environment in the form of differential equations or difference equations, or discrete models like STRIPS operators, are unavailable. An analytical design that maps the percepts of an agent to its actions (e.g., using differential game theory or control theory) in these domains is not possible without a complete model. Even if a model were available, standard methods for deriving agents are extensional and involve exploration of the entire state space. They fail because the domains considered in this paper have of the order of a 100 million states.
- **Task characteristics:** Task are sequential decision problems: payoff is obtained at the end of a sequence of actions and not after individual actions. Examples are pursuit-evasion in a single or multi-pursuer setting and navigating in a world with moving obstacles. The tasks are typically multi-objective in nature: for instance, minimize energy consumption while maximizing the time till capture by the pursuer.
- **Agent characteristics:** The agent has imperfect sensors. Imperfections occur in the form of noise, as well as incompleteness (all aspects of the state of the world cannot be sensed by our agent, a problem called *perceptual aliasing* in Whitehead and Ballard, 1990). Stochastic differential game theory has methods for deriving

agents with noisy sensors, but it requires detailed models of the noise as well as a detailed model of the environment and agent dynamics.

The action set of the agents and the values taken on by sensors are discrete and can be grouped into qualitative equivalence classes. This is the basis for the design of the parametric agent. A similar intuition underlies the design of fuzzy controllers that divide the space of sensor values into a small set of classes described by linguistic variables.

In such domains, human designers derive an initial solution by hand and use numerical methods (typically very dependent on the initial solution) to refine their solution. Our ultimate objective is to automate the derivation of good initial solutions by using general knowledge about the environment, task, and agent characteristics and thus provide a better starting point for the refinement process. We begin with the SKB and advice.

2 Compiling Advice

Our operationalization method compiles high-level domain-specific knowledge (e.g., advice) and spatial knowledge (SKB) into low-level reactive rules directly usable by the agent. The compilation performs *deductive concretion* (Michalski, 1991) because it deductively converts abstract goals and other knowledge into concrete actions. An important question is why we adopt a deductive procedure for operationalization of advice. At this time, we are able to achieve operationalization without resorting to any form of non-deductive inference. This is because for the domains studied in this paper, the SKB is complete enough to yield good parametric designs with deductive inference alone. We expect that as we expand our experimental studies to cover more domains, the

incompleteness of the SKB will force us to adopt more powerful operationalization methods.

We assume all the knowledge provided is operationalized immediately; however, it need not be applied immediately. We precompile all high-level knowledge because the agent will apply it in a time-critical situation. The learning cost prior to execution is not a concern, but the reaction time of the agent is critical. Therefore it is best to have all knowledge in a quickly usable (operational) form. Compiled rules are fully operational, whereas advice and SKB rules have at least some nonoperational elements. The user specifies what is operational for the agent.

Compilation uses two stacks: a GoalStack and an (operational condition) OpCondStack. Three types of knowledge are initially given to the compiler: facts, nonoperational rules (abbreviated *nonop rules*), and advice. A user can provide any of the three. The SKB has only facts and nonop rules. The output from compilation is a set of *op rules* directly usable (i.e., operational) by the agent. Facts have the form:

Predicate(X_1, \dots, X_n).

Nonop rules have the form:

IF cond AND ... AND cond <AND action>
THEN goal.

Anything in angle brackets is optional. The portion preceding the "THEN" is the rule *antecedent*, and the portion following the "THEN" is the rule *consequent*. A nonop rule consequent is a single goal. The syntax for a goal is "function(X_1, \dots, X_n) = value" or "predicate(X_1, \dots, X_n)". Each X_i is an object (e.g., an agent). The syntax for a "cond" (condition) or "action" in the rule antecedent is the same as for goals. Advice has the form:

<IF cond AND ... AND cond THEN>
ACHIEVE goal.

Although advice has a similar syntax to nonop rules, its interpretation differs. Advice recommends achieving the given "goal" under the given "conds". A nonop rule, on the other hand, states that the given "goal" will be achieved if the given "conds" (and "action") occur. Compilation results in stimulus-response op rules of the form:

IF cond AND ... AND cond THEN action.

The conditions of an op rule are sensor values detectable by the agent. The action can be performed by the agent. Our compilation algorithm is in Figure 1.

Push advice on GoalStack: goal followed by conditions.

Initialize OpCondStack to be empty and invoke Compile(GoalStack, OpCondStack).

Procedure Compile (GoalStack, OpCondStack)

IF GoalStack is not empty THEN

g ← pop(GoalStack);

CASE g:

1. g is an operational condition:

Push(g, OpCondStack);

Compile(GoalStack, OpCondStack)

2. g matches a fact:

Compile(GoalStack, OpCondStack)

3. g is nonoperational:

FOREACH nonop rule R_i in

knowledge base whose consequent matches g DO

Push(antecedent(R_i), GoalStack);

Compile(GoalStack, OpCondStack)¹

4. g is an operational action

Form a new op rule from the contents of OpCondStack and g;

Clear OpCondStack

ELSE Clear OpCondStack

FIG. 1. Algorithm for operationalizing advice.

¹ To prevent cycles, the last nonop rule used in step 3 is marked as "used" so that it will not be used again.

This algorithm takes advice and backchains through the SKB and user-provided nonop rules until an operational action is found. Once an operational action is found, it pops back up the levels of recursion, attaching conditions along the way, to form a new reactive agent op rule.

Let us examine a simple example of how this algorithm operates, as shown in Figures 2 and 3. *Heading*(X,Y) refers to the direction of motion of Y relative to X, and *bearing*(X,Y) refers to the direction of Y relative to X. Suppose the advice is "IF speed(adversary) = low THEN ACHIEVE heading(adversary, agent) = not(head-on)" (i.e., avoid adversary). Figure 2 shows how SKB nonop rules match this advice for backchaining, thereby creating an "and" tree. Anything preceded by a "*" is operational.

Figure 3 shows this algorithm in operation. Note that stacks grow downward. The algorithm begins by pushing the advice goal, followed by the advice condition, on the GoalStack. It then calls procedure Compile, which moves the advice condition to the OpCondStack because it is operational. The advice goal is not operational. In our example, the advice goal can be unified with the goal of SKB RULE1, which states "IF bearing(agent, adversary) = right AND turn(agent) = left THEN heading(adversary, agent) = not(head-on)". The condition and action of RULE1 are pushed on the GoalStack. Because the condition of RULE1 is operational, it is moved to the OpCondStack.

At this point, the action of RULE1 is at the top of the GoalStack, and it is operational, so we can create an op rule. The conditions from the OpCondStack are added to the action. This creates an op rule that states "IF speed(adversary) = low AND bearing(agent,

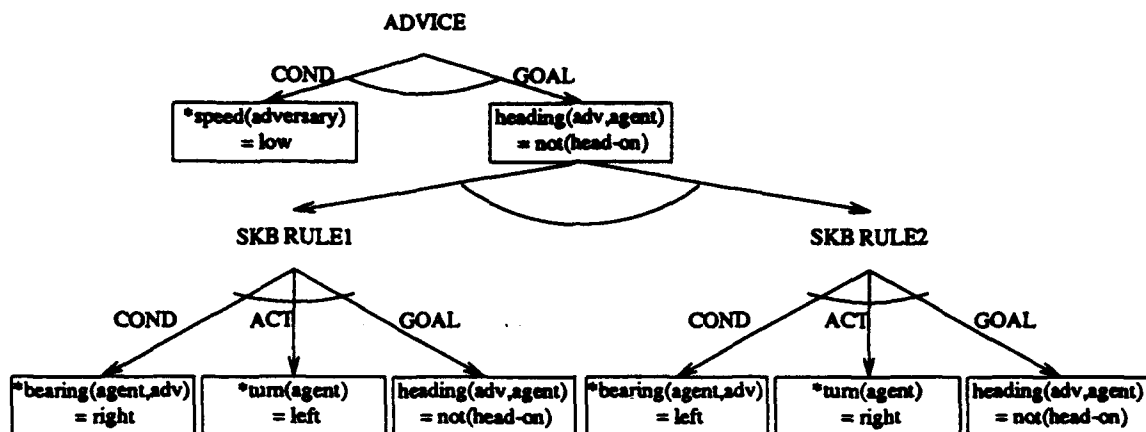


FIG. 2. Graph of example.

adversary) = right THEN turn(agent) = left". Both stacks are cleared. The algorithm continues similarly to generate a second op rule that states "IF speed(adversary) = low AND bearing(agent, adversary) = left THEN turn(agent) = right" from SKB RULE2 (see Figure 2).

Next, we apply a conversion from qualitative to quantitative op rules. The rules are given default quantitative ranges. For example, if "speed" has two values, "slow" and "fast", we bisect the range of all possible values into two subranges. Then, we allow the system to improve this initial choice of quantitative

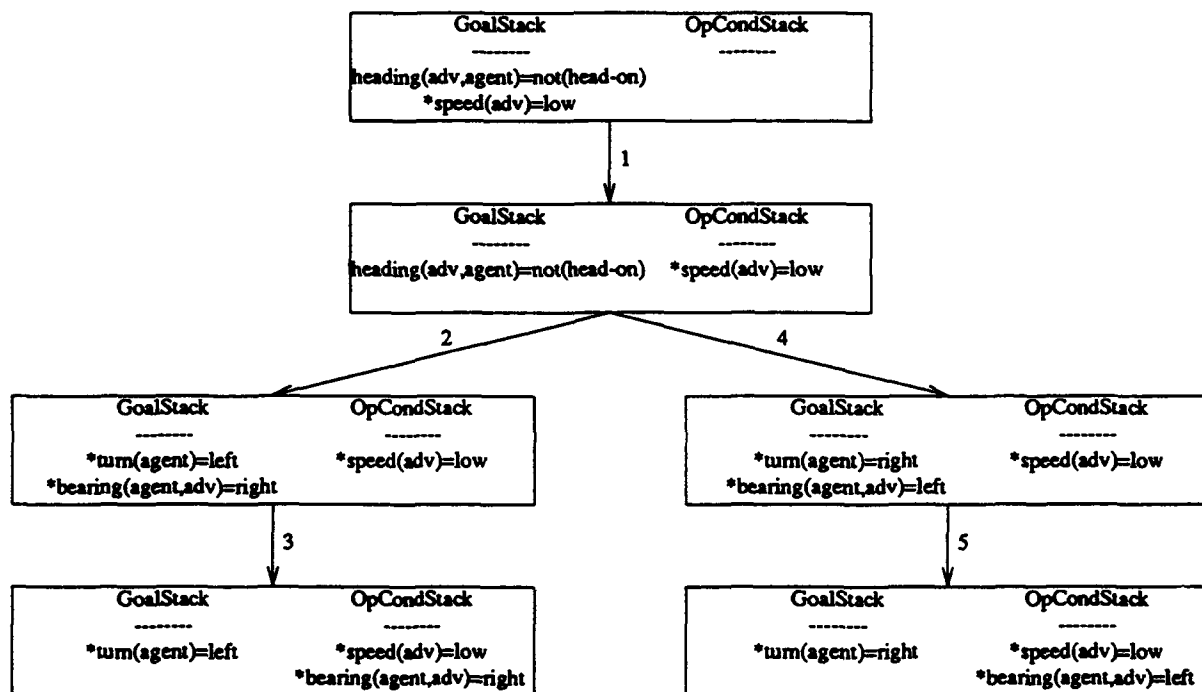


FIG. 3. Example of the compilation algorithm.

ranges by using a GA to refine the initial ranges while interacting with the environment.

3 Executing and Refining Advice

The system we use to refine and apply the op rules derived from our compiled advice is the SAMUEL reactive planner (Grefenstette *et al.*, 1990). We have chosen SAMUEL because this system has already proven to be highly effective for refining rules on complex domains (Grefenstette *et al.*, 1990; Schultz and Grefenstette, 1990). SAMUEL adopts the role of an agent in a multiagent environment in which it senses and acts. This system has two major components: a performance module and a learning module. Section 4.2 explains how performance interleaves with learning in our experiments.

The performance module, called the Competitive Production System (CPS), interacts with a simulated or real world by reading sensors, setting effector values, and receiving payoff from a critic. CPS performs matching and conflict resolution on the set of op rules. This performance module follows the match/conflict-resolution/act cycle of traditional production systems. Time is divided into *episodes*: the choice of what constitutes an episode is domain-specific. Episodes begin with random initialization and end when a critic provides payoff. At each time step within an episode, CPS selects an action using a probabilistic voting scheme based on rule strengths. All rules that match (or partially match - see Grefenstette *et al.*, 1990) the current state bid to have their actions fire. The actions of rules with higher strengths are more likely to fire. If the world is being simulated, then after an action fires, the world model is advanced one simulation step and sensor readings are updated.

CPS assigns credit to individual rules based on feedback from the critic. At the end of each episode, all rules that suggested actions taken during this episode have their strengths incrementally adjusted to reflect the current payoff. Over time, rule strengths reflect the degree of usefulness of the rules.

SAMUEL's learning module is a genetic algorithm. GAs are motivated by simplified models of heredity and evolution in the field of population genetics (Holland, 1975). GAs evolve a population of individuals over a sequence of *generations*. Each individual acts as an alternative solution to the problem at hand, and its *fitness* (i.e., potential worth as a solution) is regularly evaluated. During a generation, individuals create *offspring* (new individuals). The fitness of an individual probabilistically determines how many offspring it can have. Genetic operators, such as *crossover* and *mutation*, are applied to the offspring. Crossover combines elements of two individuals to form new individuals; mutation randomly alters elements of a single individual. In SAMUEL, an individual is a set of op rules. In addition to genetic operators, this system also applies non-genetic knowledge refinement operators, such as "generalize" and "specialize", to op rules within a rule set.

The interface between our compilation algorithm and the SAMUEL system is straightforward. The output of our compilation algorithm is a set of op rules for the SAMUEL agent. Because the op rules may be incomplete, a *random rule* is added to this rule set. The random rule recommends performing a random action under any conditions. This rule set, along with CPS and the GA learning component for improving the rules, is our initial agent.

4 Evaluation

We have not yet analyzed the cost of our compilation algorithm. The worst case cost appears to be exponential, because the STRIPS planning problem (which is P-space complete) can be reduced to it. In the future, we plan to investigate methods to reduce this cost for complex realistic problems. Potential methods include: (1) attaching a likelihood of occurrence onto advice, which enables the agent to prioritize which advice to compile first if time is limited, (2) tailoring the levels of generality and abstraction of the advice to suit the time available for compilation (e.g., less abstract advice is closer to being operational and therefore requires less compilation time), and (3) generating a parallel version of the algorithm.

We have evaluated our multistrategy approach empirically. We focus on answering the following questions:

- Will our advice compilation method be effective for a reactive agent on complex domains?
- Will the coordination of multiple learning techniques lead to improved performance over using any one learning method? In particular, we want the GA to improve the success rate of the compiled advice, and the advice to improve the convergence rate of the GA. An improved convergence rate is useful when learning time is limited.
- Can we construct a portable SKB?

4.1 Domain characteristics

To address our questions, we have run experiments on two complex problems: Evasion and Navigation. Our choice of domains is motivated by the results of Schultz and Grefenstette (1990), who have obtained large performance improvements by initializing the

GA component of SAMUEL with hand-coded op rules in these domains. Their success has inspired the work described here. Our objective is to automate their tedious manual task, and the work described here is one step toward our goal.

Both problems are two-dimensional simulations of realistic tactical problems. However, our simulations include several features that make these two problems sufficiently complex to cause difficulties for more traditional control theoretic or game theoretic approaches (Grefenstette *et al.*, 1990):

- A weak domain model. The learner has no initial model of other agents or objects in the domain. Most control theoretic and game theoretic models make worst case assumptions about adversaries. This yields poor designs in the worlds we consider because we have statistical rather than worst case adversaries.
- Incomplete state information. The sensors are discrete, which causes a many-to-one mapping and perceptual aliasing.
- A large state space. The discretization of state space makes the learning problem combinatorial. In the Evasion domain, for instance, over 25 million distinct feature vectors are observed, each requiring one of nine possible actions, giving a total of over 225 million maximally specific condition-action pairs.
- Delayed payoff. The critic only provides payoff at the end of an episode. Therefore a credit assignment scheme is required.
- Noisy sensors. Gaussian noise is added to all sensor readings. Noise consists of a random draw from a normal distribution with mean 0.0 and standard deviation equal to 5% of the legal range for the corresponding sensor. The value that results is discretized according to the

defined granularity of the sensor. A 5% noise level is sufficient to degrade SAMUEL's performance.

4.2 Experimental design

Two sets of experiments are performed on each of the two domains. Perception is noise-free for the first set, but noisy for the second. The primary purpose of the first set is to address our question about the effectiveness of our advice compilation method alone, without GA refinement. Facts, nonop rules, advice, and the random rule are given to the compiler and the output is a set of op rules. This rule set is given to SAMUEL's CPS module and applied within the simulated world model. The baseline performance with which these rules are compared is the random rule alone. These experiments measure how the average (over 1000 episodes) success rate of the compiled rules compares with that of the baseline as problem complexity increases. Statistical significance of the differences between the curves with and without advice are presented. Significance is measured using the large-sample test for the differences between two means.

The primary purpose of the second set of experiments is to address our question about the effectiveness of the multistrategy approach (compilation followed by GA refinement). Facts, nonop rules, and advice are given to the compiler and the output is a set of op rules. This rule set, plus the random rule, becomes every individual in SAMUEL's initial GA population, i.e., it seeds the GA with initial knowledge. The baseline performance with which these rules are compared is SAMUEL initialized with every individual equal to just the random rule. In either case, GA learning evolves this initial population. In other words, we compare the performance

of advice seeding the GA with GA learning alone (i.e., random seeding). *Random seeding produces an initially unbiased GA search; advice initially biases the GA search - hopefully into favorable regions of the search space.*

In this second set of experiments, performance interleaves with GA refinement. SAMUEL runs for 100 generations using a population size of 100 rule sets. Every 5 generations, the "best" (in terms of success rate) 10% of the current population are evaluated over 100 episodes to choose a single plan to represent the population. This plan is evaluated on 1000 randomly chosen episodes and the average success is calculated. This entire process is repeated 10 times and the average success rate over all 10 trials is found. The curves in our graphs plot these averages. For this set of experiments, statistical significance is measured using the two-sample *t*-test, with adjustments as required whenever the *F* statistic indicates unequal variances.

We add sensor noise, as defined in Section 4.1, for this second set of experiments because GAs can learn robustly in the presence of noise (Grefenstette *et al.*, 1990). Two performance measures are used: the success rate and the convergence rate. The convergence rate is defined as the number of GA generations required to achieve and maintain an *n*% success rate, where *n* is different for each of the two domains. The value of *n* is set empirically.

4.3 Evaluation on the Evasion problem

Our simulation of the Evasion problem is partially inspired by (Erikson and Zytkow, 1988). This problem consists of an agent, which is controlled by SAMUEL, that moves

in a two-dimensional world with a single adversary pursuing the agent. The agent's objective is to avoid contact with the adversary for a bounded length of time. Contact implies the agent is captured by the adversary. The problem is divided into episodes that begin with the adversary approaching the agent from a random direction. The adversary initially travels faster than the agent, but is less maneuverable (i.e., it has a greater turning radius). Both the agent and the adversary gradually lose speed when maneuvering, but only the adversary's loss is permanent. An episode ends when either the adversary captures the agent (failure) or the agent evades the adversary (success). At the end of each episode, a critic provides full payoff for successful evasion and partial payoff otherwise, proportional to the amount of time before the agent is captured. The strengths of op rules that fired are updated in proportion to the payoff.

The agent has the following operational sensors: time, last agent turning rate, adversary speed, adversary range, adversary bearing, and adversary heading. The agent has one operational action: it can control its own turning rate. For further detail, see (Grefenstette *et al.*, 1990).

In our experiments, we provide the following domain-specific knowledge:

FACTS

Chased_by(agent, adversary).

Moving(agent). Moving(adversary).

NONOP RULES

IF chased_by(X, Y) AND range(X, Y) = close AND turn(X) = Z THEN turn(Y) = Z.

IF range(X, Y) = not(close) AND heading(Y, X) = not(head_on) THEN avoids(X, Y).

IF turn(adversary) = hard THEN decelerates(adversary).

ADVICE

IF speed(adversary) = high THEN ACHIEVE decelerates(adversary).

IF speed(adversary) = low THEN ACHIEVE avoids(agent, adversary).

We also include knowledge of the agent's operational sensors and actions as facts.

Ten SKB nonop rules are used (they are instantiations of the two rules described in English in the introduction of this paper). Although room does not permit listing them all, some examples are:

IF bearing(X, Y) = right AND turn(X) = left THEN heading(Y, X) = not(head_on).

IF bearing(X, Y) = left AND moving(X) AND turn(X) = left THEN range(X, Y) = close.

From our input and our SKB rules, the compilation method of Section 2 generates op rules. The sensor values of these rules are translated from qualitative values to default quantitative ranges. For example, "bearing = left" is translated into "bearing = [6..12]", where the numbers correspond to a clock, e.g., 6 means "6 o'clock". Every new rule is given a strength of 1.0 (the maximum). The final op rule set includes rules such as:²

IF speed(adversary) = [700..1000] AND range(agent, adversary) = [0..750] THEN turn(agent) = hard-left.

The total number of op rules generated from our advice is 16.

We begin our experiments by addressing the first question, which concerns the

² To generate a few of these rules, we used a variant of our compilation algorithm. We omitted a description of this variation for the sake of clarity. See (Gordon and Subramanian, 1993) for details.

effectiveness of our advice-taking method. We do not use the GA. Problem difficulty is varied by adjusting a "safety" envelope around the agent. The "safety" envelope is the distance at which the adversary can be from the agent before the agent is considered captured by the adversary.

Figure 4 shows how the performance (averaged over 1000 episodes) of these op rules compares with that of just the random rule. All of the differences between the means are statistically significant (using significance level $\alpha = 0.05$). From Figure 4 we see that from difficulty levels 80 to 120, the agent is approximately twice as successful with advice than without it. This is a 100% performance advantage. Furthermore, for levels 120 to 160, the agent is about four times more effective with advice. For levels 160 to 200, the agent is an order of magnitude more effective with advice. We conclude that as the difficulty of this problem increases, the advice becomes more helpful. These results answer our first question: our advice compilation method is effective on this domain.

We address the second question about multistrategy effectiveness by combining the compiled advice with GA refinement. Figure 5 shows the results of comparing the performance of the GA with and without advice.

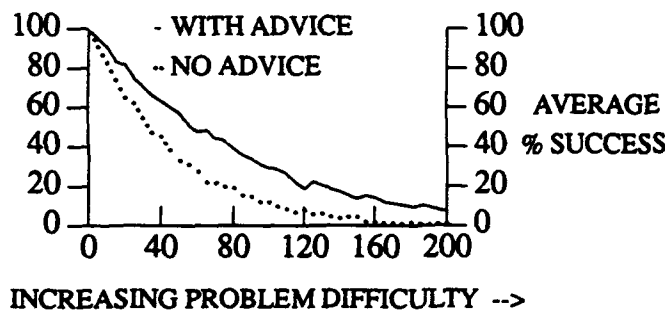


FIG. 4 Evasion domain.

The "safety" envelope is fixed at 100 (chosen arbitrarily) and noise is added to the sensors. For this domain, the convergence rate is the number of GA generations required to maintain a 60% success rate.

Figure 5 shows that in a small amount of time (less than 10 generations), the GA provides a substantial (50%) improvement in success rate. However, the convergence rate with and without advice is the same. Furthermore, although prior to 50 generations the differences between the means are not statistically significant (other than the initial boost provided by the advice), after 50 generations the improvement without advice over advice is significant ($\alpha = 0.05$). Therefore, this domain fails to demonstrate the superiority of combining strategies. We conjecture that our advice is not properly biasing the GA into the most favorable regions of the search space.

Note that these results illuminate the tight coupling that exists between the two strategies in our multistrategy system. Consistently high performance depends not only on successful compilation of advice. It also depends on how the advice initially biases the GA. A ripe area for future research is to experimentally determine effective initialization methods.

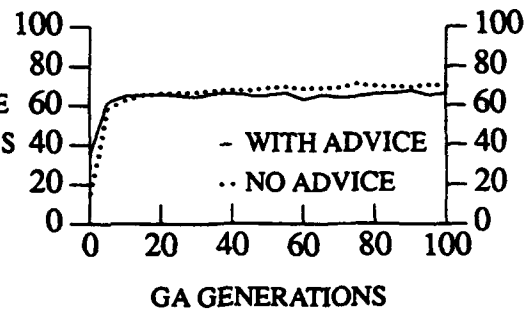


FIG. 5. Evasion domain with GA.

4.4 Evaluation on the Navigation problem

In the Navigation domain, our agent is again controlled by SAMUEL in a two-dimensional simulated world. The agent's objective is to avoid obstacles and navigate to a stationary target with which it must rendezvous before exhausting its fuel (implemented as a bounded length of time for motion). Each episode begins with the agent centered in front of a randomly generated field of obstacles with a specified density. An episode ends with either a rendezvous at the target location (success) or the exhaustion of the agent's fuel or a collision with an obstacle (failure). At the end of an episode, a critic provides full payoff if the agent reaches the target, and partial payoff otherwise, depending on the agent's distance to the goal.

The agent has the following operational sensors: time, the bearing of the target, the bearing and range of an obstacle, and the range of the target. The agent has two operational actions: it can control its own turning rate and its speed. For further detail, see Schultz and Grefenstette (1992).

We provide the following domain-specific knowledge (in addition to a list of operational sensors and actions):

FACTS

Moving(agent).

NONOP RULES

IF range(X , Y) = not(close) AND heading(Y , X) = not(head_on) THEN avoids(X , Y).

ADVICE

IF range(agent, obstacle) = not(close) THEN ACHIEVE range(agent, target) = close.

IF range(agent, obstacle) = close THEN ACHIEVE avoids(agent, obstacle).

ACHIEVE speed(agent) = high.

The same 10 SKB nonop rules used for

Evasion are again used for this domain, which confirms the portability of our SKB, thus addressing the third question. A total of 42 op rules are generated.³

Again, we address the first question by using SAMUEL without the GA. The success rate is averaged over 1000 episodes. Without advice, the average success rate is 0% because this is a very difficult domain. Figure 6 shows how we improve the success rate to as much as 90% by using our advice on this domain. At all but the last few points, the differences between the means are statistically significant ($\alpha = 0.05$). When we vary the number of obstacles, performance follows a different trend than for the Evasion domain. By far the greatest benefit of the advice occurs when there are few obstacles. Performance is 10 times better when there is only one obstacle, for example. The advantage drops as the problem complexity increases. After difficulty level 80, advice no longer offers any benefit.

Our experiments on both domains confirm that our advice compiler can be effective, however, they also indicate that the usefulness of advice may be restricted to a particular range of situations. Another learning task, which we are currently exploring, would be to identify this range and add additional conditions to the advice.

We address the second question by comparing the performance of the GA with and without advice. Noise is added. Figure 7 shows the results. All differences between the means are statistically significant ($\alpha = 0.05$). Here, the number of obstacles is fixed at five (chosen arbitrarily). For this domain,

³ We were able to decrease the number of op rules to 9 by making one careful qualitative to quantitative mapping choice.

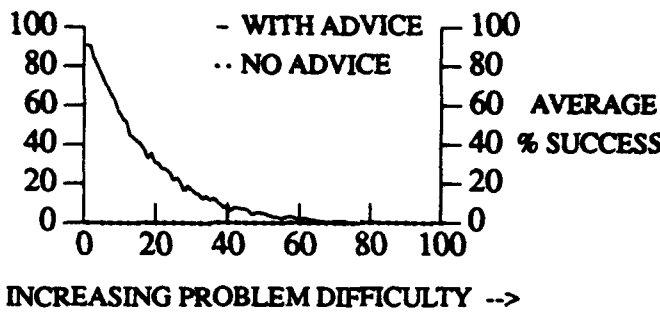


FIG. 6. Navigation domain.

the convergence rate is the number of GA generations required to maintain a 95% success rate.

Figure 7 shows that the addition of advice yields an enormous performance advantage on this domain. Figure 7 also shows that given a moderate amount of time (10 generations), the GA provides a 10% increase in the success rate. Furthermore, the addition of advice produces an 18-fold improvement in the convergence rate over using GAs alone. Not only does advice improve the convergence rate, but it also improves the level of convergence: after 80 generations, the GA with advice holds a 99% or above success rate whereas after all 100 generations the GA without advice still cannot get above a 97% success rate. For this problem, the advice appears to be biasing the GA into a very favorable region of the search space.

To further test our compilation method, we have recompiled our Navigation advice into op rules for a Nomad 200 mobile robot that is equipped with very noisy sonar and infrared sensors and can adjust its turning rate and speed. The sensors are so noisy that the robot sometimes mistakes two boxes four feet apart for a wall. The op rules that result from compilation have not been refined by the GA to develop a tolerance to noise; therefore, this

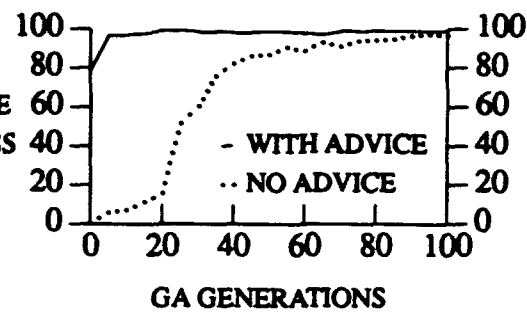


FIG. 7. Navigation domain with GA.

noise poses a severe challenge.

The op rules are linked to a vendor-provided interface that translates the language of the SAMUEL rules (e.g., "IF asonar4 [17..85] THEN SET turn -400") into joint velocity and servo motor commands. From high-level advice to avoid obstacles and rendezvous with a goal point, our method has compiled rules that enable the robot to succeed approximately a third of the time in avoiding three large boxes and reaching a goal point on the other side of a room. The same SKB rules are used for compilation. With the random rule alone, it is extremely unlikely to successfully complete this task. Our next step will be to refine these robot rules using GAs within a simulation.⁴

In conclusion, our multistrategy system offers two advantages. First, it provides an initial "boost" from seeding with initial high-level knowledge. Although this boost is insubstantial on Evasion, on Navigation we see an order of magnitude in improvement in the convergence rate. Second, the multistrategy system provides the robustness and improvement gained from GA refinement.

⁴ We wish to use SAMUEL both to handle the noise and because we had to manually refine the qualitative to quantitative mappings somewhat - SAMUEL could automate this.

Refinement yields a 10% increase in success rate on Navigation and a 50% increase on Evasion.

5 Related Work

This work relates most strongly to the following topics in machine learning: advice taking, combining projective and reactive planning, methods for compiling high-level goals into reactive rules, learning in fuzzy controllers, and multistrategy learning. This work also relates to research in differential game theory. We discuss each in turn.

5.1 Machine learning

Advice taking has been considered as early as 1958 (McCarthy) and later by Mostow (1983) and others. To date, research on assimilating advice in embedded agents has been limited but encouraging. Previous research has focused mainly on providing low-level knowledge. For example, Laird *et al.* (1990) and Clouse and Utgoff (1992) have had good success providing agents with information about which action to take. Chapman (1990) gives his agent high-level advice. Our advice taker differs from Chapman's because it can operationalize advice long before the advice is applied and because it refines the advice with a GA. Most important of all, our advice taking method is unique because it involves a multistrategy approach that couples a knowledge-intensive *deductive* precompilation phase with an empirical *inductive* refinement phase.

We assume that high-level knowledge is operationalized but not applied immediately. Methods for operationalizing advice that will be applied immediately include STRIPS-like planners (Nilsson, 1980) and explanation-based learning (EBL) planners (e.g., Segre,

1988). A closely related system is Mitchell's (1990). This system combines EBL projective planning with reactive planning. Our method for compiling goals is similar to that of EBL because it uses the notion of operability. It differs because we do not assume that the advice will be applied immediately, and therefore our compilation method has no current state on which to focus plan generation. All of the above-mentioned methods create a projective plan to achieve a goal from the current state. We precompile advice for multiple possible states.

Because our method precompiles plans from possible states rather than from a current state, it is very similar to the methods of Schoppers (1987) and Kaelbling (1988) for compiling high-level goals into low-level reactive rules. Our method differs from those of Schoppers and Kaelbling because it includes the EBL notion of operability. Also unlike Schoppers and Kaelbling, we use a refinement method following compilation.

Considerable prior work has focused on knowledge refinement. Others have used GAs to refine qualitative to quantitative mappings. For example, Karr (1991) uses GAs to select fuzzy membership functions for a fuzzy controller. Lin (1991), Mahadevan and Connell (1991), and Singh (1991) initialize their systems with modular agent architectures then refine them with reinforcement learning. Lin trains a robot by giving it advice in the form of a sequence of desired actions. Mahadevan and Connell initialize their reinforcement learner with a prespecified subsumption architecture, and Singh guides his reinforcement learner by giving it abstract actions to decompose.

One of the most similar approaches to ours is that of Towell and Shavlik (1991). They also

couple rule-based input with a refinement method; however, their refinement method is neural networks. This multistrategy system converts rules into a network topology. The content of each rule is preserved; therefore, the transformation is syntactic. Our multistrategy system, on the other hand, focuses primarily on semantic transformations that use qualitative knowledge about movements in space to convert abstract goals into concrete actions. The deductive compilation scheme (but not the refinement) is in common with Mitchell's (1987) derivation of a strategy for pushing objects in a tray using a qualitative theory of the process.

5.2 Differential game theory

Differential game theory is a branch of mathematical optimal control theory. It assumes that the behavior of the controlled system can be modeled as a system of ordinary differential equations (ODEs). The evasion problem considered in this paper is a typical example of a differential game. In particular, the problem is two-person zero-sum differential game with a *constant terminal time*. Both the pursuer and the evader move in a bounded rectangle in two dimensions. The evader has to avoid getting to within a certain distance of the pursuer for a certain length of time. In the minimax formulation of the problem, the optimal strategy of the evader is one that achieves its objective under the least favorable assumptions on the motion of the pursuer.

Differential games are formulated mathematically by specifying the motion equations of the pursuer and evader, the class of admissible controls for both systems (which identifies the way in which the pursuer and evader can change their motions), and the target or goal functional. A classic reference

for this is (Basar and Olsder, 1982). The focus of work in differential game theory is to identify conditions under which optimal strategies for the evader can be derived. This assumes complete knowledge of the dynamics of the evader and pursuer, both of which are unavailable to us. The theory would be more useful to us if it had a qualitative counterpart which allowed us to determine the existence of solutions to the evader's problem from partial knowledge of the evader and pursuer's dynamics.

6 Discussion

We have presented a novel multistrategy learning method for operationalizing and refining high-level advice into low-level rules to be used by a reactive agent. Operationalization uses a portable SKB. An implementation of this method has been tested on two complex domains and a Nomad 200 robot.

We have learned the following lessons:

(1) Our advice compiler can be effective on complex domains, and it will be important to identify the regions of greatest effectiveness for advice, (2) A portable SKB appears feasible, and (3) Coordinating a deductive learning strategy (advice compilation) with an inductive learning strategy (GA refinement) can lead to a substantial performance improvement over either method alone. This success, however, depends on how the advice biases the GA search. Future work will focus on identifying those characteristics of advice that bias this search favorably. We will also focus on further addressing our questions about performance using different advice and alternative domains (e.g., Subramanian and Hunter, 1993).

Many other interesting directions are suggested by our experimental results. At present we do not consider the cost of

incorporating advice. For larger scale problems and situations where advice is provided more frequently, the agent has to reason about the costs and benefits of compiling advice at a given point in time. Classical issues in trading off deliberation time for action time are relevant here. We have chosen the GA method for refinement because it was readily available to us. A comparison of neural network refinement schemes and reinforcement learning schemes on the problems studied here will provide valuable insights into the tradeoffs between various refinement strategies. We believe that multistrategy learning systems of the future must have a bank of operationalization and refinement methods at their disposal and have fast methods for selecting them. We have chosen a specific breakdown of effort between the advice compilation and refinement phases. How this coordinates with our choice of problem domains and refinement schemes is another question for future study.

Acknowledgements

We appreciate the helpful comments and suggestions provided by Bill Spears, Alan Schultz, Connie Ramsey, John Grefenstette, Alan Meyrowitz, and the anonymous reviewers.

References

- Basar, T. and G. Olsder, *Dynamic Non-cooperative Game Theory*. New York: Academic Press, 1982.
- Chapman, D., *Vision, instruction and action*. Ph.D. thesis. MIT, 1990.
- Clouse, J. and P. Utgoff, A teaching method for reinforcement learning. In *Proc. of the Ninth International Workshop on Machine Learning*, 1992.
- Erickson, M. and J. Zytkow, Utilizing experience for improving the tactical manager. In *Proc. of the Fifth International Workshop on Machine Learning*, 1988.
- Gordon, D. and D. Subramanian, Assimilating advice in embedded agents. Unpublished manuscript, 1993.
- Grefenstette, J., Ramsey, C., and A. Schultz, Learning sequential decision rules using simulation models and competition. *Machine Learning*, Volume 5, Number 4, 1990.
- Holland, J. *Adaptation in Natural and Artificial Systems*. University of Michigan Press: Ann Arbor, 1975.
- Kaelbling, L., Goals as parallel program specifications. In *Proc. of the Seventh National Conference on Artificial Intelligence*, 1988.
- Karr, C., Design of an adaptive fuzzy logic controller using a genetic algorithm. In *Proc. of the Ninth International Conference on Genetic Algorithms*, 1991.
- Laird, J., Hucka, M., Yager, E., and C. Tuck, Correcting and extending domain knowledge using outside guidance. In *Proc. of the Seventh International Conference on Machine Learning*, 1990.
- Lin, L., Programming robots using reinforcement learning and teaching. In *Proc. of the Ninth National Conference on Artificial Intelligence*, 1991.
- Maes, P. and R. Brooks, Learning to coordinate behaviors. In *Proc. of the Eighth*

National Conference on Artificial Intelligence, 1990.

Mahadevan, S. and J. Connell, Automatic programming of behavior-based robots using reinforcement learning. In *Proc. of the Ninth National Conference on Artificial Intelligence*, 1991.

McCarthy, J., Mechanisation of thought processes. In *Proc. Symposium*, Volume 1, 1958.

Michalski, R., Inferential learning theory as a basis for multistrategy task-adaptive learning. In *Proc. of the Eighth International Workshop on Multistrategy Learning*, 1991.

Mitchell, T., Becoming increasingly reactive. In *Proc. of the Eighth National Conference on Artificial Intelligence*, 1990.

Mitchell, T., Mason, M., and A. Christiansen, Toward a learning robot. CMU Technical Report, 1987.

Mostow, D. J., Machine transformation of advice into a heuristic search procedure. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 1). Tioga Publishing Co., Palo Alto, CA., 1983.

Nilsson, N., *Principles of Artificial Intelligence*. Tioga: Palo Alto, 1980.

Odetayo, M. and D. McGregor, Genetic algorithm for inducing control rules for a dynamic system. In *Proc. of the Third International Conference on Genetic Algorithms*, 1989.

Schoppers, M., Universal plans for reactive robots in unpredictable environments. In

Proc. of the Sixth National Conference on Artificial Intelligence, 1987.

Schultz, A. and J. Grefenstette, Using a genetic algorithm to learn behaviors for autonomous vehicles. In *Proc. of the Navigation and Control Conference*, 1992.

Schultz, A. and J. Grefenstette, Improving tactical plans with genetic algorithms. In *Proc. of the IEEE Conference on Tools for AI*, 1990.

Segre, A., *Machine Learning of Robot Assembly Plans*. Kluwer: Boston, 1988.

Singh, S., Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proc. of the Ninth International Workshop on Machine Learning*, 1992.

Subramanian, D. and S. Hunter, Some preliminary studies in agent design in simulated environments. Unpublished manuscript, 1993.

Tesauro, G., Temporal difference learning of backgammon strategy. In *Proc. of the Ninth International Workshop on Machine Learning*, 1992.

Towell, G. and J. Shavlik, Refining symbolic knowledge using neural networks. In *Proc. of the Eighth International Workshop on Multistrategy Learning*, 1991.

Whitehead, S. and D. Ballard, Learning to perceive and act by trial and error. *Machine Learning*, Volume 7, Number 1, 1991.

REGAL: An Integrated System for Learning Relations Using Genetic Algorithms

A. Giordana and L. Saitta
Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149 Torino (Italy)

Abstract

Inducing concept descriptions from examples requires a large space of hypotheses to be explored. Genetic algorithms offer an appealing alternative to traditional search algorithms, because of their multi-point search strategy. In this paper, the new system REGAL is described: it uses genetic algorithms to learn first order logic concept descriptions. Moreover, it can be easily integrated with a deductive component, in order to exploit a domain theory. Two approaches to learning disjunctive concept descriptions are presented: the first one is a modification of the classical method of learning one disjunct at a time, whereas the second one is based on the idea of fitness sharing and tries to let subpopulations be spontaneously formed, according to the theory of the niches and species. The approaches have been compared on an artificial domain.

Key words: Learning relations, Genetic algorithms

1. Introduction

Inducing concept descriptions from examples and background knowledge is a fundamental machine learning task which can be formulated as a search problem (Mitchell,

1982; Michalski, 1983). What distinguishes an approach from another is the concept description language, the search method and the hypothesis selection criterion.

Genetic algorithms offer a powerful, domain-independent search method: they have been first used in machine learning associated to the "classifier" model (Holland, 1986), but, recently, they have also been used for concept induction, both in propositional calculus (De Jong & Spears, 1991; Vafaie & De Jong, 1991; Bala et al., 1991; Janikov, 1992) and in first order logic (Giordana & Sale, 1992). From these first experiments, genetic algorithms proved to be an appealing alternative for traditional search algorithms, because of their great exploration power, useful to escape local minima, and their suitability to exploit massive parallelism.

This paper extends in many respects the framework presented in (Giordana & Sale, 1992) in order to overcome some limitations of that approach. In particular, the algorithm GA-SMART, presented there, could only learn concepts described in conjunctive normal form, with no explicit negation. In the present extension, multi-modal concepts, described in a more powerful first order logic language, can be learned; in this language negation of single atoms and negation of existentially quantified formulas may occur.

The resulting inductive algorithm can be easily integrated into a deductive/inductive paradigm such as the one described by Bergadano and Giordana (1988).

2. System Overview

The system REGAL, described in this paper, is an evolution of GA-SMART (Giordana & Sale, 1992) and adopts the same method of encoding first order logic (FOL) formulas into bit strings. GA-SMART was a straightforward evolution of the *Simple Genetic Algorithm* proposed by De Jong (1975).

Two alternative approaches are proposed in the literature in order to encode the "problem solutions" handled by genetic algorithms. The first one suggests a fixed-length bit string representation. In this way, the genetic algorithm architecture becomes problem independent and a substantial amount of investigation, available in the literature (Goldberg, 1989a), can be exploited to design the genetic operators. The drawback is that it may be difficult to formulate a problem in such a way that it can be encoded using this representation. The second approach prefers an encoding scheme more closely fitting the original formulation of the problem. The drawback is that the genetic operators must be designed *ad hoc*. For concept induction, a variable-length bit string representation has been used in (De Jong & Spears, 1991) and a tree-like representation in (Janikov, 1991). In both cases, special genetic operators and control strategies have been designed.

In GA-SMART and REGAL we chose the first approach and, hence, we devoted our efforts to the proper encoding of FOL formulas into fixed-length bit strings. This has been achieved by restricting the hypothesis

representation language \mathcal{L} in such a way that it becomes finite. The complexity of \mathcal{L} is defined through a *language template* Λ , which represents the maximally complex formula in \mathcal{L} . Any other well formed formula is obtained by deleting some literal from Λ .

In GA-SMART, formulas in \mathcal{L} were in conjunctive normal form and negation was not explicitly allowed. The richer language used in REGAL allows the system to learn the same set of formulas as other systems do (Bergadano et al., 1988, 1991; Quinlan, 1990; Pazzani & Kibler, 1992). Another important advantage of REGAL is its suitability to be integrated with a deduction system and used to refine, by induction, inconsistent concept descriptions generated by EBG (Mitchell, 1986) with an incomplete and/or inconsistent domain theory. Finally, a further improvement introduced in REGAL is the use of a new kind of sharing functions (Goldberg & Richardson, 1987), allowing the formation of sub-populations. Crowding was already used in GA-SMART in order to learn many concepts at the same time. Here, the technique is modified in order to learn concept descriptions expressed as disjunctions of Horn clauses.

REGAL's learning strategy consists in searching for a maximum value of the fitness function $f(\varphi)$ associated to a formula $\varphi \in \mathcal{L}$. The algorithm starts with a set $A(0)$ of formulas (individuals) randomly selected in \mathcal{L} . Each individual $\varphi_i \in A(0)$ is associated to the corresponding fitness value $f(\varphi_i)$. In order to explore new points in the search space, the population $A(0)$ is let evolving by applying five operators: *reproduction*, *mating*, *crossover*, *mutation* and *seeding*. Let $A(t)$ be the solution population at time t ; reproduction consists in selecting a multiset $A'(t)$ of

individuals by sampling the elements of $A(t)$ with probability proportional to the corresponding value of f . Then, in the mating phase, the elements of $A'(t)$ are randomly paired, and each pair undergoes crossover with probability p_c , i.e., new offsprings are created by recombining in some way the information encoded in the parents. The new individuals partially or totally replace $A(t)$, obtaining a new generation $A(t+1)$. Basically, the algorithm searches for the highest values of f by statistically recombining tentative solutions already discovered. As this strategy could cause a loss of essential information, the *seeding* operator (in addition to classical mutation) is exploited to introduce new information in the evolution process.

As mentioned in Section 1, the basic scheme described above has been made a little more sophisticated, by incorporating the mechanisms of *crowding* (De Jong, 1975) and of *sharing functions* (Goldberg & Richardson, 1987) in order to allow the formation of subpopulations. This turns out to be useful when many concepts are to be learned at the same time or when concepts have a multimodal structure, which requires disjunctive descriptions to be learned.

REGAL can work serially as well as in parallel on a multi-processor according to the network model described by Goldberg (1989b). A discussion of the parallel implementation on a 16 transputers network can be found in (Giordana & Sale, 1992). The results presented in this paper, have been obtained using a single processor workstation Sun 10.

3. Knowledge Representation and Encoding

The language L , used by REGAL, is a first order logic language containing conjunction, disjunction, negation, internal disjunction (Michalski, 1983) and existential and universal quantification. The building blocks of the language are "internally disjunct" atoms, such as, for instance:

$$\text{colour}(x, \text{yellow} \vee \text{green}) \quad (3.1)$$

in which one of the arguments of a predicate is a disjunction of constants. Formula (3.1) is semantically equivalent to:

$$\text{colour}(x, \text{yellow}) \vee \text{colour}(x, \text{green}) \quad (3.2)$$

but is more compact and readable.

More in general, a predicate P of arity m is specified using the following syntax:

$$P(x_1, x_2, \dots, x_m, [v_1, v_2, \dots, v_n]) \quad (3.3)$$

where the complex term $[v_1, v_2, \dots, v_n]$ denotes the maximal internal disjunction, i.e., the set of all the possible values the feature P can assume on the tuple of variables $\langle x_1, x_2, \dots, x_m \rangle$. Any other disjunction inside P is represented by a subset of the set $[v_1, v_2, \dots, v_n]$. If the set $[v_1, v_2, \dots, v_n]$ does not exhaust the possible values of P , this set is completed by means of the symbol $*$. Then, in the formula $P(x_1, x_2, \dots, x_m, [v_1, v_2, \dots, v_n, *])$ the symbol $*$ denotes "no one of the values v_1, v_2, \dots, v_n ". In other words, $*$ is an abbreviation for the expression $\neg v_1 \wedge \neg v_2 \wedge \dots \wedge \neg v_n$. Therefore, the symbol $*$ allows a restricted form of negation to be implemented. As an example, let us consider the predicate $\text{colour}(x, [\text{yellow}, \text{green}, \text{blue}, \text{grey}, *])$. Then, $\text{colour}(x, [\text{yellow}, *])$ is equivalent to $\text{colour}(x, \text{yellow} \vee [\neg \text{yellow} \wedge \neg \text{green} \wedge \neg$

blue $\wedge \neg$ gray]) or, equivalently, to \neg colour(x, green) $\wedge \neg$ colour(x, blue) $\wedge \neg$ colour(x, grey).

Deleting a term from an internal disjunction is a specialization operation. For instance¹ :

$$P(x, [v_1]) \prec P(x, [v_1, *]) \prec P(x, [v_1, v_2, *])$$

In particular, a predicate is said in maximally specific form (*msf*) when its internal disjunction contains only one term, and in maximally general form (*mgf*), when its internal disjunction contains all the possible values, including “*”. We notice that, owing to the completeness hypothesis, a predicate with an empty internal disjunction has to be considered illegal and one in *mgf* is tautologically true.

A second form of negation, greatly increasing the power of the language \mathcal{L} , is the negation of existentially quantified formulas. This form of negation, widely used in logic programming, can be learned by systems such as ML-SMART (Bergadano et al. 1988, 1991), FOIL (Quinlan, 1990, 1991) and FOCL (Pazzani & Kibler, 1992). A negated existentially quantified formula has, in \mathcal{L} , the following syntax:

$$\neg \exists y_1, \dots, y_m [\psi(x_1, \dots, x_n, y_1, \dots, y_m)] \quad (3.4)$$

where ψ is a disjunction of (possibly internally disjunct) predicates, each one containing at least one variable in the set y_1, y_2, \dots, y_m . The genetic operators can deal with negated formulas in a similar way as they do with positive ones.

¹ The relation “ ψ is more general than φ ” is denoted by $\varphi \prec \psi$, according to Michalski [1983].

Deleting a term from the internal disjunction of a predicate occurring in ψ is a generalization operation, as it appears from the equivalence of formulas (3.5) and (3.6):

$$\neg \exists x [(P(x, [a \vee b]) \vee (Q(x, [e \vee f]))) \quad (3.5)$$

$$\forall x [\neg P(x, a) \wedge \neg P(x, b) \wedge \neg Q(x, a) \wedge \neg Q(x, b)] \quad (3.6)$$

Finally, the occurrence inside ψ of a predicate in *mgf* leads to an *absurdum*, owing to the completeness hypothesis, whereas a predicate with an empty internal disjunction leads to a tautology.

The knowledge base acquired by REGAL consists of a flat set of concept descriptions in disjunctive normal form:

$$\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n \rightarrow h \quad (3.7)$$

where h denotes a concept and the φ_i 's ($1 \leq i \leq n$) are conjunctions of predicates, possibly containing, in turn, internal disjunctions. In the following we will describe the language template and the method for mapping each conjunction φ_i to a fixed-length bit string.

3.1. The Language Template

As mentioned in Section 2, the language \mathcal{L} is characterized by a language template Λ , which is a conjunctive formula such that every other conjunctive formula φ_i of \mathcal{L} can be obtained by deleting some literal from Λ ². In REGAL the template Λ obeys the following syntax:

$$\Lambda \equiv \varphi(x_1, \dots, x_n) \wedge \neg \exists y_1, \dots, y_m [\psi(x_1, \dots, x_n, y_1, \dots, y_m)] \quad (3.8)$$

² In this paper “literal” is used to refer to a single constant in an internal disjunction, because this last can be transformed into a disjunction of literals, as shown by (3.1) and (3.2)

In (3.8) both ϕ and ψ denote conjunctions of predicates; moreover, each predicate in ψ must contain at least one variable in the set y_1, y_2, \dots, y_m , as in (3.4). The template Λ is, then, partitioned into two subformulas, Λ^+ and Λ^- , corresponding to the positive and negated parts of Λ , respectively. In Fig. 3.1 an example of a language template is reported, for the sake of illustration.

$$\Lambda \equiv \text{colour}(x, [\text{red}, \text{blue}, *]) \wedge \\ \text{shape}(x, [\text{square}, \text{triangle}, *]) \wedge \\ \neg \exists y [\text{colour}(y, [\text{red}, \text{blue}, *]) \wedge \\ \text{far}(x, y, [0, 1, 2, 3, *])]$$

$$\Lambda^+ \equiv \text{colour}(x, [\text{red}, \text{blue}, *]) \wedge \\ \text{shape}(x, [\text{square}, \text{triangle}, *])$$

$$\Lambda^- \equiv \neg \exists y [\text{colour}(y, [\text{red}, \text{blue}, *]) \wedge \\ \text{far}(x, y, [0, 1, 2, 3, *])]$$

Fig. 3.1 – Example of a language template including the unary predicates “colour” and “shape” and the binary predicate “far”. The template describes a scene in which no object y , of any color, is at any distance from an object x of any color and shape.

Any predicate occurring in ϕ or ψ is in *mgf*. Each ϕ_i , occurring in any concept description (3.7), is a particular instantiation of Λ . For example, the formula:

$$\text{colour}(x, [\text{red}]) \wedge \text{shape}(x, [\text{square}]) \wedge \quad (3.9) \\ \neg \exists y [\text{colour}(y, [\text{blue}]) \wedge \text{far}(x, y, [2, 3])]$$

is an instantiation of the template reported in Fig. 3.1. Formula (3.9) describes a scene in which no blue object has a distance value of 2 or 3 from a red square.

As discussed before, each predicate in Λ^+ is tautologically true, and, thus, Λ^+ is also tautologically true. On the contrary, Λ^- is

tautologically false and, hence, Λ is tautologically false. As a consequence, a predicate in *mgf* can be deleted from the positive part of a formula without changing its extension, whereas a predicate with an empty internal disjunction can be deleted from the negated part of a formula without changing its extension.

In the language \mathcal{L} there may also exist predicates which do not have internal disjunction (for instance, $\text{equal}(x, y)$). These predicates are suggested and added to the language template by the background knowledge, but are not considered during the inductive learning process, because they represent *necessary constraints* that must always be present in every concept description and are not processed by the generalisation mechanism of REGAL.

3.2. Mapping Formulas to Bit Strings

The language template can be represented using a bit string $s(\Lambda)$, where each literal (term) occurring in the maximal internal disjunction of a predicate in Λ is associated to a corresponding bit in $s(\Lambda)$. Predicates in Λ which do not have internal disjunction (necessary constraints) do not need to be associated to any bit in the string.

By keeping adjacent in $s(\Lambda)$ the bits corresponding to literals that are adjacent in the template, $s(\Lambda)$ will be partitioned into two parts $s(\Lambda^+)$ and $s(\Lambda^-)$, corresponding to Λ^+ and Λ^- , respectively. The bit string associated to the template of Fig. 3.1 is reported in Fig. 3.2. Any other formula in \mathcal{L} , obtained by deleting some literal from Λ , can be represented by the bit string $s(\Lambda)$, in which the values of the bits have been properly set.

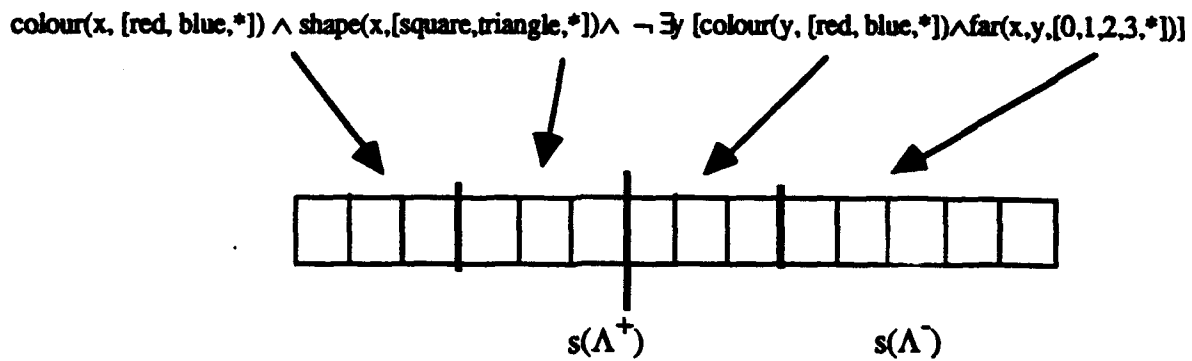


Fig. 3.2 – Bit string associated to the language template reported in Fig. 3.1

This last point needs a separate discussion for the positive and negative parts of the template.

The semantic interpretation of the alleles in the bit string has been defined on the basis of the previous considerations. In particular, for the positive part of a formula, if the bit corresponding to a given term v in a predicate P is set to 1, then v belongs to the current internal disjunction of P , whereas if it is set to 0, it does not belong to it. Hence, a substring containing all 0's for a predicate is illegal and it is automatically rewritten as a string of all 1's. On the contrary, for the negated part of a formula, the semantic interpretation is the opposite: setting to 1 a bit corresponding to a given term v means that v is absent from the corresponding internal disjunction, whereas setting it to 0 means that it is present. Again, a substring containing all 0's for a predicate is

illegal, whereas a substring containing all 1's corresponds to the maximal generality for a conjunct. Then, the system again replaces a string of all 0's with a string containing all 1's, whenever it occurs.

In Fig. 3.3 the bit string corresponding to the formula of Fig. 3.2 is reported.

3.3. Integrating REGAL in a Deductive Learning Framework

Several authors suggested to use an inductive procedure to refine concept descriptions obtained by EBG with imperfect domain theories. In (Bergadano & Giordana, 1988), for instance, tentative concept descriptions are generated from non-operational ones, using a possibly incomplete and/or inconsistent domain theory and a set of learning events.

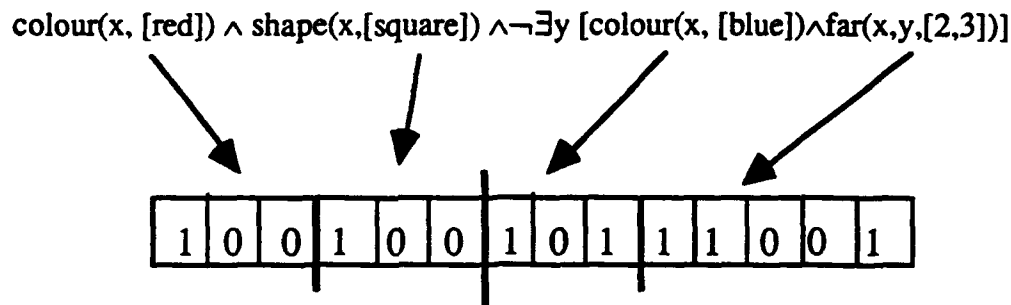


Fig. 3.3 – Bit string corresponding to the formula reported in Fig. 3.2.

The deductive engine, that performs EBG with many examples at the same time, is similar to those used in deductive databases. In this way, those concept descriptions, which need to be refined because they are inconsistent, can be detected immediately. The inductive procedure is similar to the one that was used later in FOIL (Quinlan, 1990).

An improvement of the method described in (Bergadano & Giordana, 1988) was proposed later by the same authors (Bergadano et al., 1989). The inductive refinement of a concept description can be made more effective by telling the system which parts of the domain theory are incomplete. This is done by extending the classical Horn clause language with a special construct called "predicate set", which is represented using the following syntax:

$$\{P_1, P_2, \dots, P_n\} \rightarrow Q \quad (3.10)$$

where P_1, P_2, \dots, P_n and Q denote predicates with their variables. Expression (3.10) means that a definition for Q was unknown and that it must be found by induction using the description language defined by the predicates occurring in the set $\{P_1, P_2, \dots, P_n\}$. Predicate sets are recognized by the deductive engine and are left in proofs as unresolved literals. Then, the inductive procedure removes the possible inconsistencies in the proofs by performing a search in the space defined by the predicate sets occurring there; as a last step, predicate sets are eliminated from the concept descriptions.

In this paper, we propose to use the internal disjunction formalism to describe predicate sets. The advantages of this choice is that the

deductive engine of ML-SMART (Bergadano et al., 1988, 1991) can be used to generate the template Λ from a domain theory. Afterwards, REGAL can be invoked as an inductive procedure to refine concept descriptions.

4. The Fitness Function

The fitness function gives an evaluation of how well a formula ϕ describes a concept h . Three main criteria are usually adopted to evaluate the quality of a concept description: consistency, completeness and simplicity (Michalski, 1983). The same criteria are adopted here and are combined in the fitness function. In the following, a family of empirical fitness functions, new with respect to the one used in GA-SMART, is presented and analysed.

Let F be the set of learning examples, and let ϕ be a candidate description of the concept h ; let, moreover, $M^+(h)$ and $M^-(h)$ denote the numbers of positive and negative instances of h in F , respectively. Finally, let $m^+(\phi)$ and $m^-(\phi)$ be the numbers of positive and negative instances of h , respectively, belonging to F and verifying ϕ .

As a measure of completeness the ratio $x = m^+(\phi)/M^+(h)$ is used, whereas the consistency is evaluated by $w = m^+(\phi)/[m^+(\phi) + m^-(\phi)]$, i.e., as the ratio between the number of positive instances and the global number of instances verifying ϕ (Bergadano et al., 1988, 1991). However, w tends to give a too optimistic evaluation, especially when the number of available negative instances is

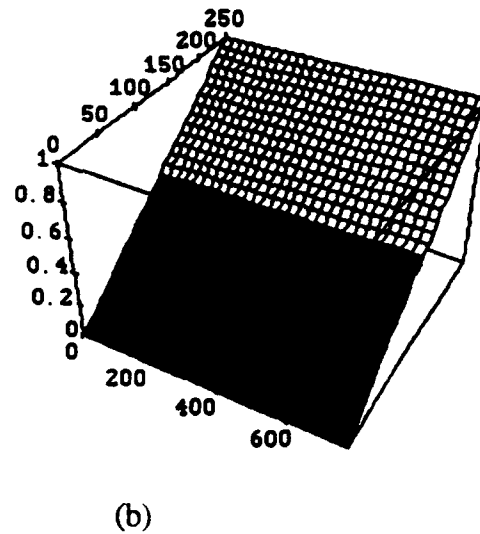
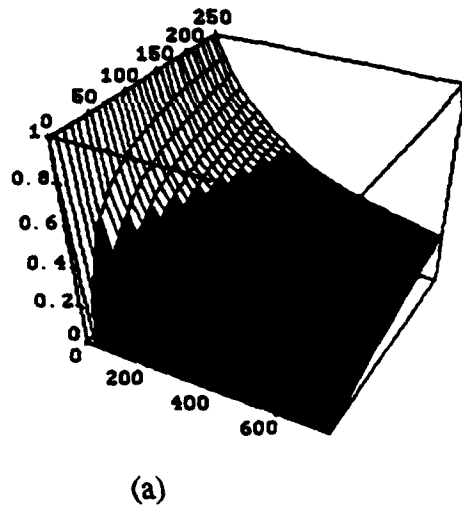


Fig. 4.1 – (a) Plot of $(1-w)$ versus $m^+(\varphi)$ and $m^-(\varphi)$. **(b)** Plot of $m^-(\varphi)/M^-(h)$ versus $m^+(\varphi)$ and $m^-(\varphi)$. The values $M^+(h) = 750$ and $M^-(h) = 250$ have been used.

small in comparison to that of positive ones. Suppose, for instance, that $M^+(h) = 750$ and $M^-(h) = 250$; a formula covering all positive instances and all negative instances will have $w = 0.75$, even if it is totally useless.

Therefore, we looked for a measure more severely penalising inconsistency. The currently adopted measure is:

$$y = \text{Max} \left\{ 1-w, \frac{m^-(\varphi)}{M^-(\varphi)} \right\} \quad (4.1)$$

In Fig. 4.1(a) and 4.1(b) the plots of $(1-w)$ and $m^-(\varphi)/M^-(h)$ versus $m^+(\varphi)$ and $m^-(\varphi)$ are reported. The plot of y versus $m^+(\varphi)$ and $m^-(\varphi)$ is reported in Fig. 4.2.

Finally, the simplicity of a formula is equated to its syntactic generality and is measured by $z = n(1)/n(s)$, where $n(1)$ is the number of 1's in a string $s(\Lambda)$, and $n(s)$ is the total number of bits in $s(\Lambda)$. This may seem a simplistic evaluation, but proved to work well in several test cases.

The three measures introduced above are combined into a unique function $f(\varphi)$, defined as follows:

$$f(\varphi) = x^\alpha (1 - y^\beta) + A (e^{B \cdot x \cdot z} - 1) + D \quad (4.2)$$

where $A, D \ll 1$ and $B, \alpha, \beta < 1$ are user-defined parameters. The resulting surface representing $f(\varphi)$ in a domain with $M^-(h) = 250$ and $M^+(h) = 750$ is reported in Fig. 4.3. We notice that the actual syntactic definition of the function f is not fundamental. What is important is the qualitative shape of the corresponding surface. The proposed function is the result of a series of trials; its computation time is irrelevant with respect to the matching time required to evaluate $m^+(\varphi)$ and $m^-(\varphi)$. The small value D has been added for the following reason: when the population is randomly initialized, it is possible that x be zero, thus making $f(\varphi)$ zero, which hinders φ from being selected for reproduction. A value of $f(\varphi)$ different from zero, even if small,

gives to ϕ a chance of being selected by the reproduction operator.

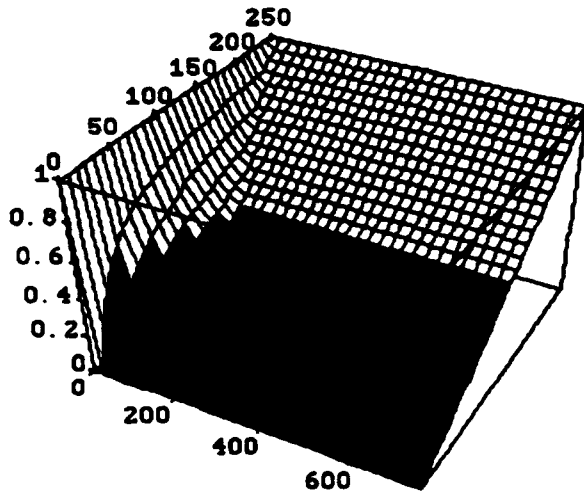


Fig. 4.2 – Plot of the y function, defined by (4.1), versus $m^+(\phi)$ and $m^-(\phi)$, with $M^+(h) = 750$ and $M^-(h) = 250$.

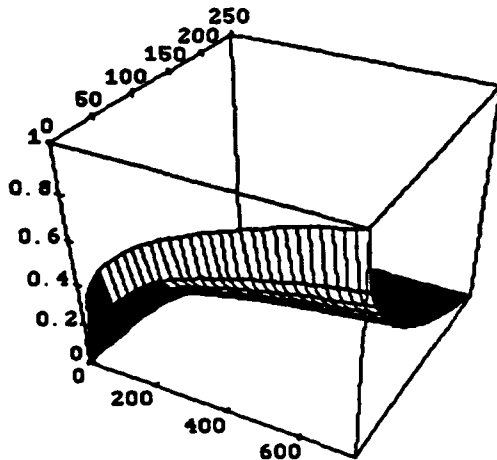


Fig. 4.3 – Plot of the fitness function $f(\phi)$, defined by (4.2), vs. $m^+(\phi)$ and $m^-(\phi)$, with $M^+(h) = 750$ and $M^-(h) = 250$. The values used for the parameters are: $\alpha = \beta = 0.2$, $A = 0.002$, $B = 0.4$ and $D = 0.000001$.

5. Genetic Operators

The fundamental genetic operators used by REGAL are the classical ones used in the literature, with the addition of two non-standard crossovers and a new form of mutation called *seeding*, which will be described later.

Crossover operators are inherited by GA-SMART; in particular, they are the *two-point crossover* and the *uniform crossover*, previously used in the literature (De Jong, 1975; Syswerda, 1989), and the *generalising* and *specialising crossovers*, specifically designed for the task at hand. As described by Goldberg (1989a), the two-point crossover creates two new offsprings by exchanging two corresponding substrings randomly selected in the parents. In the uniform crossover, the information is exchanged between the two parents in such a way to give the same chance of permutation to every bit position. The parent strings s_1 and s_2 are scanned from left to right and a probability $p = 0.5$ of exchanging the values is given to each pair of corresponding bits. The two crossovers have been selected empirically after experimenting with several data sets.

The generalising and specialising crossovers need additional explanations. As described in Fig. 3.2, the string $s(\Lambda)$ is divided into substrings, each one corresponding to a specific predicate P_i . In both crossover types, a set \mathcal{D} of predicates is randomly selected in Λ . The specialising crossover works as follows: (a) The substrings in s_1 and s_2 , corresponding to the predicates not selected in \mathcal{D} , are copied unchanged into the corresponding offsprings s'_1 and s'_2 . (b) For each predicate $P_i \in \mathcal{D}$, a new substring s'_i is generated by AND-ing the

bits of the corresponding substring $s_1(P_i)$ and $s_2(P_i)$. The substring s'_i is then copied in both s'_1 and s'_2 . The generalising crossover differs from the specialising one in that the new predicate s'_i is obtained by OR-ing the corresponding bits of s_1 and s_2 . The set \mathbf{D} is created by assigning an equal probability p_d of being selected to each predicate P_i .

Given a pair of strings $\langle s_1, s_2 \rangle$, generated by the mating procedure, crossover will be applied with an assigned probability p_c . Then, the specific crossover type is selected statistically by taking into account the features of s_1 and s_2 . The probabilities p_u of uniform crossover, p_{2pt} of two-point crossover, p_s of specialising crossover and p_g of generalising crossover are assigned through the following set of functions:

$$\begin{aligned} p_u &= (1 - a \cdot f) \cdot b \\ p_{2pt} &= (1 - a \cdot f) \cdot (1 - b) \\ p_s &= a \cdot f \cdot r \\ p_g &= a \cdot f \cdot (1 - r) \end{aligned} \quad (5.1)$$

In equations (5.1), a and b are tunable parameters, $f = [f(s_1) + f(s_2)]/2$ is the mean value of the fitness of the two strings s_1 and s_2 , and $r = [(m^+(s_1) + m^-(s_1) + m^+(s_2) + m^-(s_2)) / [(M^+ + M^-) \cdot 2]]$ is the mean value of the ratio between the number of instances covered by ϕ_1 and ϕ_2 , respectively, and the global number of instances in the training set, where ϕ_1 and ϕ_2 are the formulas associated to the bit strings s_1 and s_2 . When a formula does not cover any instance, then $r = 0$ and specialisation is never applied; when a formula covers all the instances, then $r = 1$ and generalisation is never applied.

The *seeding operator* is primarily used to initialise the population in order to start with a set of formulas covering at least some examples in F . In fact, the concept description

language characterised by a template can be so large that a randomly generated population may have no one individual matching any element in F . In this case, the fitness is close to zero for all individuals and the search reduces to a random walk for a long initial phase, until formulas covering a few examples are discovered by chance. The seeding operator receives in input a string s (corresponding to a formula ϕ) and returns a modified string s' , which covers at least one instance randomly selected from F . Therefore, the action performed by this operator resembles the selection of the seed in the Star methodology (Michalski, 1983; Gemello et al., 1991). Finally, seeding operator can also be used by REGAL, in alternative to classical mutation, in order to reintegrate genetic information lost during the evolution of the program. This point will be discussed further in the next section.

6. Learning Multimodal Concepts

The fundamental novelty of REGAL is its ability to learn disjunctive concepts. Learning disjunctive concepts is a problem inherently deceptive for a genetic algorithm. In fact, each separate disjunct in the concept description corresponds to a local maximum of the fitness function. Therefore, the genetic algorithm will frequently try to apply crossover between individuals representative of different disjuncts, creating thus offsprings that are necessarily worst than the parents (because a consistent common generalization does not exist).

Two strategies have been experimented in order to deal with this problem. The first one is an adaptation of a commonly used technique, which suggests to learn a disjunct at a time (Michalski, 1980; Bergadano et al., 1988; Quinlan, 1990). The second one, based

on the theory of the niches and species (Deb & Goldberg, 1987), tries to learn a set of complete and consistent disjunctive descriptions by encouraging the formation of subpopulations. Both techniques will be discussed and compared using an artificial domain, where a difficult multimodal concept has been constructed.

6.1. Learning one Disjunct at a Time

The test application has been designed by extending the well known train set used by Michalski (1980). Also in the present case, we have two concepts to distinguish: trains going East and trains going West. Therefore, each learning event is represented as a sequence of items (coaches), each one described by a vector of attributes referring to shape, colour, position, length, number of wheels and number of loads. Thousands of trains have been generated by a program which selects at random the values of the attributes. Then, each train has been classified using a set of disjunctive rules. The challenge for REGAL was to discover the original rules or a set of equivalent ones.

The rules for classifying trains going East (the first concept) are reported in the following:

Class 1 - Trains going East

Rule 1: In second position there is an open-top small coach, carrying one load, followed by an open-top small coach.

Rule 2: In third position there is a closed-top small coach carrying one load and an open top small coach in fifth position.

Rule 3: In position two, three or four there is a small coach, with two wheels and carrying one load, immediately followed by a long white coach carrying one load.

The learning set used for the experiment contained 500 instances of Class 1 and 500 instances of Class 2. Rule 1 covered 98 instances of Class 1, Rule 2 covered 206 and Rule 3 covered 209. The three subsets were slightly overlapping, because 13 instances verified more than one rule.

The concept description language used by REGAL is very similar to the one described in (Michalski, 1980) and is reported in Table I.

Using the strategy of learning a disjunct at a time, REGAL was able to solve the problem. In particular, it learned a complete and consistent description of Class 1, consisting of three disjuncts, covering 216, 200 and 90 positive examples and roughly corresponding to Rule 3, 2 and 1, respectively. For example, the description of the largest disjunct (216 examples), is the following:

```
coach(x) ∧ coach(y) ∧ follows(x,y) ∧
length(x, [1]) ∧ Nload(x, [1]) ∧
length(y,[2]) ∧ Nload(y, [1,3]) ∧
colour(y, [white])
```

These results have been obtained using a population of 800 individuals initialized by the seeding operator. The genetic evolution has been controlled using a linear fitness scaling mechanism (Goldberg, 1989a) and a generation gap of 35%, i.e., only about one third of the individuals was replaced at each generation.

The system was ran repeatedly, in order to find a single partial definition each time, until all the training instances of the target concept were covered. In particular, at each run, the system was let free to converge to some

Table I
 Predicates and template characterizing the concept description language used by REGAL for learning
 the concept of trains going East

<u>Predicates mapped into the bit string</u>	<u>Comment</u>
Position(x, [0, 1, 2, 3, 4]) starting from the engine (position 0)	The position number of the coach,
Length(x, [1, 2])	Length of the coach (Short or Long)
Wheels(x, [2, 3])	Number of wheels
Nloads(x, [0, 1, 2, *])	Number of loads
Colour(x, [yl, wh, rd, gr, gy, bk])	yl = yellow, wh = white, etc.
Shape(x, [ot, en, us, or, cr, el, jt, h, sl])	ot = open-top, en = engine, etc.
Distant(x,y, [0, 1, *])	Number of coaches between x and y
<u>Constraint predicates</u>	<u>Comment</u>
Follow(x,y)	Item y follows item x
Coach(x)	Item x is a coach
<u>Template</u>	
Coach(x) \wedge Position(x, [0, 1, 2, 3, 4]) \wedge Length(x, [1, 2]) \wedge Wheels(x, [2, 3]) \wedge Nloads(x, [0, 1, 2, *]) \wedge Colour(x, [yl, wh, rd, gr, gy, bk]) \wedge Shape(x, [ot, en, us, or, cr, el, jt, he, sl]) \wedge	
Coach(y) \wedge Position(y, [0, 1, 2, 3, 4]) \wedge Length(y, [1, 2]) \wedge Wheels(y, [2, 3]) \wedge Nloads(y, [0, 1, 2, *]) \wedge Colour(y, [yl, wh, rd, gr, gy, bk]) \wedge Shape(y, [ot, en, us, or, cr, el, jt, he, sl]) \wedge	
Position(z, [0, 1, 2, 3, 4]) \wedge Length(z, [1, 2]) \wedge Wheels(z, [2, 3]) \wedge Nloads(z, [0, 1, 2, *]) \wedge Colour(z, [yl, wh, rd, gr, gy, bk]) \wedge Shape(z, [ot, en, us, or, cr, el, jt, he, sl]) \wedge	
Follow(x,y) \wedge Distant(x,y, [0, 1, *]) \wedge Distant(y,z, [0, 1, *]) \wedge Distant(x, z, [0, 1, *])	

concept definition ϕ , which, of course, covered only one subset ϕ^* of the positive instances of Class 1. Then, the instances in ϕ^* were removed from F and the system was restarted. Learning one disjunct at a time is probably the most easy way to cope with the deceptiveness of the problem. In fact, as deception is due to the simultaneous presence of competing disjuncts, removing them as soon as they are discovered make easier the task of learning the remaining ones.

6.2. Learning Many Disjuncts at One Time

Two approaches are proposed in the literature to ease the formation of subpopulations: *crowding* (De Jong, 1975) and use of *sharing*

functions (Goldberg & Richardson, 1987). Crowding is a variant of the basic algorithm with respect to reproduction and replacement of older individuals. In particular, the new individuals, generated by crossover and mutation, replace the older ones that are most similar to them, according to a given similarity measure. In this way, sub-populations are likely to grow up because genetic pressure tends to manifest itself primarily among similar individuals. Both in GA-SMART and in REGAL this method proved to work well to learn many concepts at one time, but was unable to allow a stable formation of subpopulations, representative of disjunctive definitions of the same concept. In all the experiments performed, in the long term there was a disjunct overcoming the other ones. The

interpretation we give, in terms of the deceptiveness of the problem, is that stronger disjuncts inhibit the reproduction of the other ones by means of unfruitful matings.

The method based on sharing functions, unlike crowding, tries to act on the reproduction probability in order to inhibit the excessive growth of the genetic pressure of a subpopulation. This is done by reducing the fitness of an individual, depending on the number of existing individuals similar to it. In the initial formulation, proposed by Goldberg & Richardson (1987), genotypical sharing was considered. The fitness value $f(\phi)$, associated to an individual ϕ , was considered as a reward from the environment to be shared with other individuals, proportionally to their similarity degree with ϕ . Similarity between two individuals ϕ and ϕ' was evaluated as the Hamming distance $d(s, s')$ between the corresponding bit strings s and s' .

However, in many cases it is better to consider a semantic distance, i.e., the phenotypical distance, rather than the syntactic one, as it has been discussed in (Deb & Goldberg, 1989). For instance, by referring to our problem, it is easy to find formulas apparently similar but having a very different extension of the learning set F . Therefore, we tried to design a proper mechanism for sharing fitness, being the one described in (Goldberg & Richardson, 1987) not suitable to our task. The philosophy underlying the sharing function approach is that subpopulations (species) live by exploiting environmental niches. If a species proliferates too much, it will be limited by the

implicit reduction of the pro-capite incoming from the niche it exploits.

In REGAL, learning events are considered as life sources that are exploited by the formulas covering them. A formula ϕ matching $m^+(\phi)$ positive events takes its support from them in order to evaluate its fitness. However, if the same events are matched also by other formulas, the fitness of ϕ is consequently reduced, because ϕ is not essential to cover such events. In this way, the reproduction rate decreases when formulas become too redundant.

The algorithm used to evaluate the fitness, shared by competing formulas, can be easily understood using the following metaphor: concept instances are cakes and formulas are living being eating cakes:

- 1) After crossover, mutation and seeding, formulas are evaluated using the function described in Section 4 for computing their absolute fitness.
- 2) Formulas, sorted according to their absolute fitness, are allowed to eat cakes. Each one takes one serving from each one of the cakes associated to the positive instances it covers. If all servings have been already eaten, it will not have any.
- 3) For each formula ϕ the shared fitness $f_{sh}(\phi)$ is evaluated according to the following expression:

$$f_{sh}(\phi) = f(\phi) E/m^+(\phi) \quad (6.1)$$

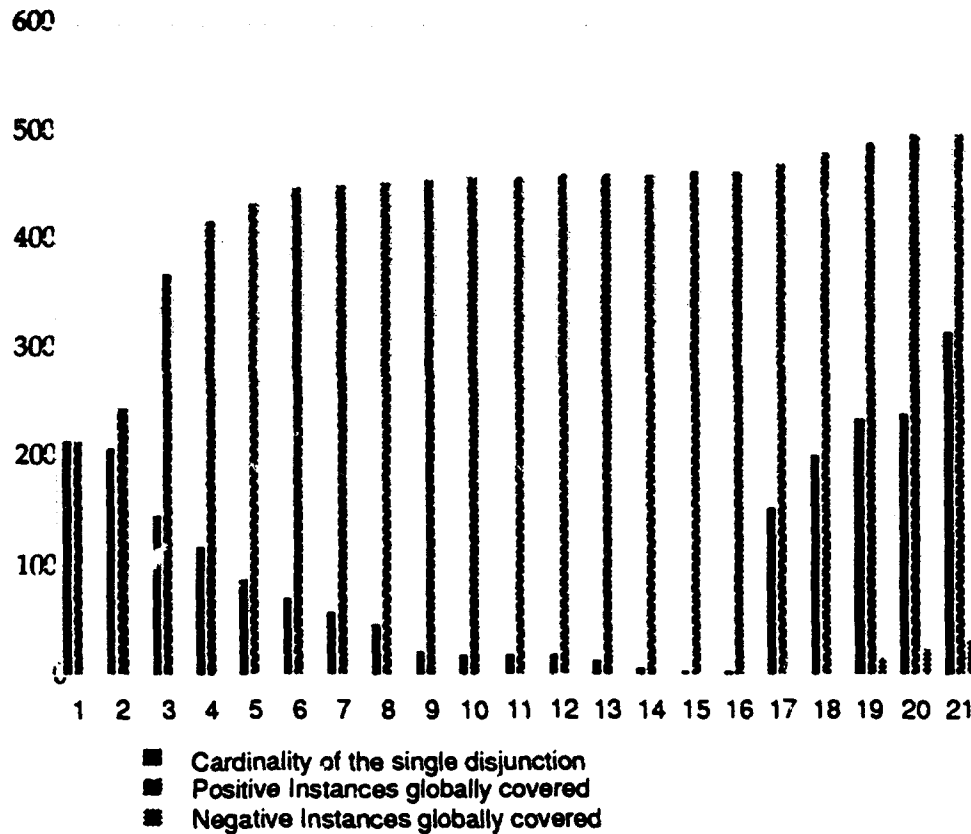


Fig. 6.1 - Results obtained using the fitness scaling method with a population of 800 individuals, using a crossover probability $p_c = 0.5$ and a generation gap of 35%.

where E represents the total amount of serving eaten by ϕ and $m^+(\phi)$ the number of positive instances covered by ϕ .

If an individual cannot eat at all, it will have a shared fitness equal to zero and then will not reproduce.

Experimentation with the test case described above showed that the method was able to control reproduction rate in order to allow the formation of stable subpopulations. The results obtained running REGAL through 200 generations and with a global population of 800 individuals, are reported in Fig. 6.1. Black columns in the histogram represent the numbers of positive instances covered by each one of the first 21 disjuncts, sorted as follows: first, the consistent disjuncts (1-16) are sorted

according to decreasing completeness and, then, inconsistent disjuncts (17-21) are sorted according to increasing inconsistency. Dashed (dotted) columns report the global number of positive (negative) instances covered by all the disjuncts from the first until the current one.

It is worth noting that the first six disjuncts cover 450 of the 500 positive instances of Class 1. They are in a static equilibrium, being positioned on F in such a way that each one exploits a good amount of instances without competitors. Such disjuncts were present in the population since the 60th generation. On the contrary, the 12 disjuncts from 9 to 21 were in a kind of dynamic equilibrium, being in hard competition for survival. They some time disappeared, to reappear later, when regenerated by the genetic evolution. In

particular, the presence of the small disjuncts from 10 to 16 is due to the continuous creation of the seeding operator.

We notice that several large disjuncts are present, some of them corresponding approximately to the definitions given by Rules 2 and 3; others were discovered by performing alternative kinds of generalization. For instance, the first disjunct corresponds to the following definition obtainable by generalising Rule 3:

$$\text{coach}(x) \wedge \text{length}(x, [1]) \wedge \text{load}(y, [1,3]) \wedge \text{colour}(y, [\text{wh}]) \wedge \text{follow}(x,y) \quad (6.2)$$

Formula (6.2) is consistent with the data and covers 216 positive instances instead of 206. This generalization has been made possible because the learning set was not large enough to be representative of all the possible cases. Increasing the number of instances of the second class, this overgeneralisation would disappear.

On the other hand, we notice the lack of a unique disjunct corresponding to the 98 instances classified by Rule 1. We explain this fact as an effect of the deceptive action of the large disjuncts which interact negatively with the growth of smaller alternative concept definitions.

7. Conclusions

In this paper we presented an extension of a method described in (Giordana & Sale, 1992) for learning concept descriptions in first order logic, using an inductive engine based on a genetic algorithm. Several substantial improvements have been introduced. First, the concept description language has been extended in order to include internal disjunction and negation. Second, we have

shown how this genetic learning paradigm can be integrated with a deductive module in the very same way as in (Bergadano & Giordana, 1988).

Two techniques have been introduced to learn disjunctive concepts. The first one learns one disjunct at a time, whereas the second one allows subpopulations to be formed. Even if the work in this direction is still in an early stage, we have presented a sharing mechanism which proved effective in allowing stable subpopulations, corresponding to different disjuncts, to be formed.

References

- Bala J., De Jong K.A. and Pachowicz P., "Learning Noise Tolerant Classification Procedures by Integrating Inductive Learning and Genetic Algorithms", *Proc. First International Workshop on Multistrategy Learning* (Harpers Ferry, WV), pp. 316-323, 1991.
- Bergadano F. and Giordana A., "A Knowledge Intensive Approach to Concept Induction", *Proc. 5th Machine Learning Conference* (Ann Arbor, MI), pp. 305-317, 1988.
- Bergadano F., Giordana A. and Ponsero S., "Deduction in Top-Down Inductive Learning", *Proc. 6th Machine Learning Workshop* (Ithaca, NY), pp. 23-25, 1989.
- Bergadano F., Giordana A. and Saitta L., "Automated Concept Acquisition in Noisy Environments". *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *PAMI-10*, 555-578, 1988.
- Bergadano F., Giordana A. and Saitta L., *Machine Learning: An Integrated Approach and its Application*, Ellis Horwood, Chichester, UK, 1991.
- Deb K. and Goldberg D., "An Investigation of Niche and Species Formation in Genetic Function Optimization", *Proc. 3rd Int. Conf.*

on *Genetic Algorithms*, Fairfax, VA, pp. 42-50, 1989.

De Jong K.A., "Analysis of the Behaviour of a Class of Genetic Adaptive Systems", *Doctoral Dissertation*, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1975.

De Jong K.A. and Spears W.M., "Learning Concept Classification Rules Using Genetic Algorithms", *Proc. IJCAI-91*, Sidney, Australia, pp. 651-656, 1991.

Gemello R., Mana F. and Saitta L., "RIGEL: An Inductive Learning System", *Machine Learning*, 6, 7-36, 1991.

Giordana A. and Sale C., "Genetic Algorithms for Learning Relations", *Proc. 9th Int. Conf. on Machine Learning*, Aberdeen, Scotland, pp. 169-178, 1992.

Goldberg D.E. *Genetic Algorithms*, Addison-Wesley, 1989a.

Goldberg D.E., "Sizing Populations for Serial and Parallel Genetic Algorithms", *Proc. 3rd Int. Conf. on Genetic Algorithms*, Fairfax, VA, pp. 70-79, 1989b.

Goldberg D.E. and Richardson J., "Genetic Algorithms with Sharing for Multimodal Function Optimization", *Proc. 2nd Int. Conf. on Genetic Algorithms*, Cambridge, MA, pp. 41-49, 1987.

Janikov C.Z., "A New System for Inductive Learning in Attribute-Based Spaces", *Lecture Notes in Artificial Intelligence*, 542, 378-388, 1991.

Holland J.H., "Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems". In R. Michalski, J. Carbonell & T. Mitchell (Eds.), *Machine Learning: An AI Approach*, Vol. II. Morgan Kaufmann, Los Altos, CA, pp. 593-623, 1986.

Michalski R., "Pattern Recognition as Rule-Guided Inductive Inference", *IEEE Transactions on Pattern Analysis and*

Machine Intelligence, *PAMI-2*, 349-361, 1980.

Michalski R., "A Theory and Methodology of Inductive Learning". In R. Michalski, J. Carbonell & T. Mitchell (Eds.), *Machine Learning: An AI Approach*, Vol. I. Morgan Kaufmann, Los Altos, CA, pp. 83-134, 1983.

Mitchell T. "Generalization as Search", *Artificial Intelligence*, 18, 203-226, 1982.

Mitchell T, Keller R.M., Kedar-Cabelli S., "Explanation-Based Generalization: A Unifying View", *Machine Learning*, 1, 47-80, 1986.

Pazzani M. and Kibler D., "The Utility of Knowledge in Inductive Learning", *Machine Learning*, 2, 57-94, 1992.

Quinlan J.R., "Learning Logical Definitions from Relations", *Machine Learning*, 2, 239-266, 1990.

Vafaie H. and De Jong K.A., "Improving the Performance of Rule Induction System Using Genetic Algorithms", *Proceedings First International Workshop on Multistrategy Learning*, Harpers Ferry, VA, pp. 305-315, 1991.

Incremental Genetic Programming and Neural Net Learning: A Case Study

Hugo de Garis

Brain Builder Group,
Evolutionary Systems Department,
ATR Human Information Processing
Research Laboratories,
2-2 Hikari-dai, Seika-cho, Soraku-gun,
Kansai Science City,
Kyoto, 619-02, Japan.
tel : + 81 7749 5 1079, fax : + 81 7749 5 1408,
email : degaris@hip.atr.co.jp

Abstract

This paper investigates whether an incremental approach to Genetic Programming (i.e. using Genetic Algorithms to build/evolve complex systems) (de Garis 1990, 1993) is possible. The vehicle used to explore this question is that of a simple mapping of binary input vectors to binary output vectors. Since this mapping is learned using Neural Networks, Genetic Algorithms, and Incrementalism - Incremental Genetic Programming (or Incremental Evolution) can be considered a form of Multistrategy Learning.

Keywords : Multistrategy Learning, Incremental Evolution, Genetic Programming (GP), Incremental GP, Genetic Algorithms (GAs), GenNets (Genetically Programmed Neural Network Modules), Artificial Nervous Systems, Biots (Biological Robots), Darwinian Robotics, 1000-GenNet Biots, GenNet Accelerators, GenNet Shaping, Cellular Automata (CAs), CA Networks, Neurite Networks, CA Neurons, CA Machines, Darwin Machines, CREEPER.

1. Introduction

This paper takes a first step in the direction of what the author calls "Incremental Evolution", and situates it in the context of Multistrategy Learning. A growing number of people around the world (especially those working in the field of Artificial Life) are now realizing that the rise of ULSI (ultra large scale integrated) circuits, and future molecular scale technologies will probably necessitate an evolutionary approach to complex system building, rather than the traditional approach of human design (blueprinting). The evolutionary building of complex systems has been labeled "Genetic Programming" by the author (de Garis 1990, 1993). Within our lifetimes, there will be so many components in systems, that they will not be designable, because these systems will have become too complex (too many components, too many complex non linear interactions). Artificial brains and artificial embryos are such examples. Such systems will have to self assemble and be evolved to overcome the complexity barrier. (The beauty of GP, is that the internal complexity of the system which is successfully evolved, is irrelevant. This is because the Genetic

Algorithm, which is the underlying tool of GP, does not care about the complexity of the systems it evolves, so long as the fitness values of the evolving systems keep increasing).

In the Brain Builder Group at ATR, we hope to build/evolve artificial brains using Genetic Programming techniques in special hardware known as Darwin Machines. Thinking about how to do this led the author directly to the problem discussed in this paper, namely - "Is it possible to do Genetic Programming incrementally?" This question will probably become very important in the next few years, because if one evolves a given system S1, which has a given level of complexity and functionality (e.g. a robot "kitten" with an artificial brain, giving it 100 "behaviors"), and one then wishes at some later time to evolve a more sophisticated system S2 (e.g. a robot "cat" with 1000 "behaviors"), does one just throw away the first system S1 and all the many years of work that went into it, or is it possible to evolve S2 using S1 as a base, i.e. can one evolve S2 incrementally from S1 (as happens in nature). The author believes that in the 1990s, there will be strong economic pressure to solve the "incremental GP problem".

Having stressed the importance of the question (i.e. can one GP incrementally), this paper makes an initial attempt at providing an answer, by taking a multistrategy learning approach to the problem of mapping binary input vectors to binary output vectors. More specifically, it shows how a multistrategy learning approach was applied to the task of generating an associative memory in an incremental fashion. The particular task chosen to illustrate this multistrategy approach is merely an illustration. The emphasis of this paper is on "*incremental evolution*", and not upon neural networks, nor associative memory, nor neural network learning algorithms which add neurons to the net incrementally. "*Incremental evolution*" is a new topic, and has little to do with the already substantial literature on what is conventionally called "incremental learning" (i.e. the incremental generation of classes, given one-at-a-time presentation of elements to be classified), despite the similarity of the two topic labels. The simple example chosen here

to illustrate the possibility or otherwise of incremental GP, was to generate a 1:1 mapping between binary input and binary output vectors, where the input/output vector pairs were not all supplied at once. Three strategies were combined to perform this learning task. The first strategy employed was to use a neural network as the vehicle to generate the association. The second strategy was to train the neural network using an evolutionary approach (i.e. using a Genetic Algorithm). The third strategy was to perform this evolution in an incremental manner. Thus this paper introduces the "*Incremental Evolution Problem*" and how a multistrategy learning approach can help solve it. More concretely, a Genetic Algorithm was employed to evolve the weight values of a fully connected neural network (called a "GenNet" (de Garis 1990, 1993) which initially contained N neurons to perform T tasks. The results (i.e. the evolved weights of the N neurons) were then taken, and to this neural network were added a few more neurons dN, to evolve the performance of a few more tasks dT. This paper investigates

- (a) whether this can be done at all (the most important question in view of the above discussion on incremental evolution, i.e. incremental GP),
- (b) and if so, as a possible bonus, whether it might be faster to evolve an N+dN GenNet performing T+dT tasks incrementally (i.e. [N,T], then [N+dN, T+dT]), than to do N+dN from scratch.
- (c) how the two approaches (i.e. from scratch or incremental) compare in task performance quality.

It is believed by the author that the concept of *Incremental Evolution* will become increasingly important as more research groups attempt to build artificial nervous systems (ANS) using evolved neural networks as modules. This type of work is now going on in at least four labs (as far as the author is aware) around the world, i.e. the author's "Brain Builder Group" at ATR, Beer's group at Case Western Reserve University, Arbib's group at the University of Southern California, and Cliff et al's group at Sussex University. Sooner or later, all of these groups will have to face the question of

Incremental Evolution, i.e. "Whether it is better (i.e. easier, quicker) to scrap an earlier, simpler artificial nervous system (ANS), by building/evolving a newer, bigger ANS from scratch, OR, whether it is better to add components to the already existing ANS, i.e. to build/evolve incrementally?" Nature was forced to build incrementally because it did not have the luxury to scrap an earlier design. Each step in nature's evolutionary path had to be from one viable design to another (incrementally modified) but equally viable design.

2. The Experiments

In an attempt to answer the three questions above (i.e. (a), (b), (c)), a GenNet of 12 (fully connected) neurons was evolved (using a Genetic Algorithm) which mapped 4-bit input vectors to 4-bit output vectors. The four input neurons were distinct from the four output neurons. During all of these experiments, the input/output ([I]/[O]) pairs that were used were taken from the following 5 :- ([1010]/[0110], [1110]/[0010], [0011]/[0100], [1100]/[1001] and [1101]/[0011]). Real input or output values less than 0.2 were arbitrarily interpreted to be a binary "0", and input or output values greater than 0.4 were arbitrarily interpreted to be a binary "1". To measure the fitness of an evolving GenNet, the following approach was used. For a given binary input vector, e.g. [1010], its desired or target output was [0110], according to the above list. The input vector [1010] and desired output vector were converted into their respective input and output signal values, to become [0.5, 0.1, 0.5, 0.1] and [0.0, 0.7, 0.7, 0.0]. (The 0.5 and 0.7 values were chosen because experience showed they facilitated GenNet dynamics). The four input vector component values were interpreted to be the (clamped) external input values of the four input neurons of the GenNet being evolved. If the outputs at the 100th cycle (giving plenty of time for transients to die out and the outputs to stabilize) were [0.23, 0.35, 0.56, -0.18], then the fitness of this output was defined to be :-

$$\begin{aligned} \text{fitness} = & 20,000 - 1000 \cdot (0.23 - 0.0)^2 \\ & - 1000 \cdot (0.7 - 0.35)^2 + (0.56 - 0.7)^2 \\ & + (0.0 - -0.18)^2 \end{aligned}$$

More formally, where v_{targ} is the desired or target output value, v_{out} is the actual output value, and dF is the fitness contribution :-

$$\begin{aligned} \text{IF } v_{\text{targ}} = 0.0 \\ \{ \text{THEN } [\text{IF } v_{\text{out}} < 0.0, \\ \quad dF = + (0.0 - v_{\text{out}})^2] \\ \text{ELSE } [\text{IF } v_{\text{out}} > 0.0, \\ \quad dF = -1000 \cdot (v_{\text{out}} - 0.0)^2]] \end{aligned}$$

$$\begin{aligned} \text{IF } v_{\text{targ}} = 0.7 \\ \{ \text{THEN } [\text{IF } v_{\text{out}} < 0.7, \\ \quad dF = -1000 \cdot (0.7 - v_{\text{out}})^2] \\ \text{ELSE } [\text{IF } v_{\text{out}} > 0.7, \\ \quad dF = + (v_{\text{out}} - 0.7)^2]] \end{aligned}$$

These fitness contributions were defined so that v_{out} values lying outside the "0" and "1" regions were heavily penalized, and so that v_{out} values lying inside these regions were rewarded mildly, so as to push the v_{out} values towards +1.0 and -1.0 (corresponding to binary values "1" and "0" respectively). If P input vectors were presented to the GenNet, the total fitness definition was defined to be :-

$$\text{fitness} = \frac{1}{P} \left(\sum_{j=1}^P (20000 + \sum_{i=1}^4 dF_{ij}) \right)$$

where i ranged over the four components of a single output vector, and j ranged over the P input vectors. A brief description of the evolution of a GenNet is now given, for those not already familiar with it.

This description has appeared already in many publications (e.g. de Garis 1990, 1993). A GenNet is a Genetically Programmed Neural Network. Genetic Programming is defined to be the use of Genetic Algorithms (GAs) as builders of complex systems. A GA was used to evolve the weight values and signs of neural connections so that the outputs of the GenNet took desired values (or in some experiments, controlled some system in desired ways). A GenNet is a fully connected network.

For a connection, and hence weight (w_{ij}) between neuron "i" and neuron "j", one bit is

used for the weight's sign (0 for an excitatory synapse, 1 for an inhibitory synapse), and 6 to 8 bits are used for the weight (assumed to have an absolute value less than 1.0, and which is expressed as a binary fraction). Thus if the number of bits "B" used to express the weight is 6, then the binary string 1110100 is equivalent to the weight -0.8125. Hence for a GenNet of N neurons, the number of bits in the chromosome which specifies all the weights and signs of the connections between the N neurons, will be $N*N*(B+1)$. Each sign and weight is concatenated onto the chromosome.

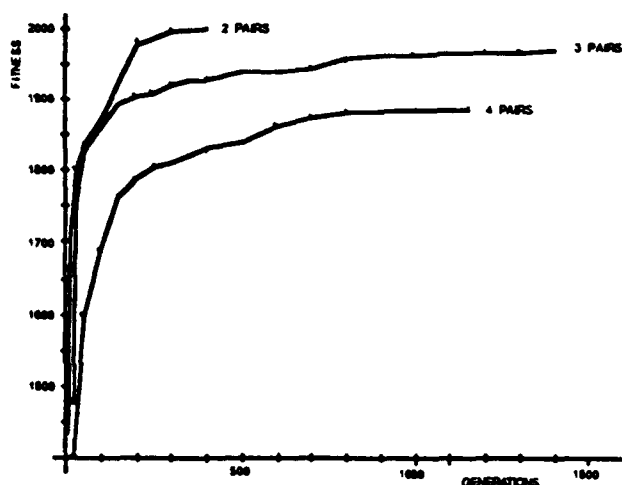


FIG.1 FITNESS EVOLUTIONS
for 12-Neuron GenNets (2,3,4 PAIRS)

Thus the connection (sign and weight) between neuron "i" and neuron "j" will be expressed by the $(i*N + j)$ th group of $(B+1)$ bits. These $(B+1)$ bits on the chromosome are called a "slot". An initial population of randomly generated chromosomes of this length is used to evolve the required GenNets. We turn now to some initial results. FIG.1 shows the fitness rise as a function of the number of generations, for the three cases of 2, 3 and 4 input/output pairs, as listed earlier. In the 2 and 3 pair cases, the final fitness values stabilized at values thought to be "acceptable", where "acceptable" was defined to mean that if the target output was 0.0, then any output less than 0.2 was passable, and if the target output was 0.7, then any output greater than 0.4 was

passable. The output values at the 100th cycle of the elite chromosomes are shown in FIG.2. The 4 input/output pair case, shows that for a 12-Neuron GenNet, 4 pairs are too many. Note that for each of these 3 graphs the 2, 3 and 4 I/O pair case GenNets were evolved from scratch, i.e. there was no incremental evolution used. A 16-Neuron GenNet was then evolved with 4 and 5 I/O pairs respectively. The 4 I/O pair case evolved "acceptably", but the 5 I/O pair case did not. See FIG.3 for the fitness growths of the 4 and 5 I/O pair cases.

TARG	OUTS	TARG	OUTS	TARG	OUTS
0.0	0.07	0.0	-0.10	0.0	0.03
0.7	0.72	0.7	0.68	0.7	0.66
0.7	0.68	0.7	0.72	0.7	0.45
0.0	-0.04	0.0	-0.52	0.0	0.00
0.0	-0.60	0.0	-0.39	0.0	0.10
0.0	-0.01	0.0	0.03	0.0	0.29
0.7	0.85	0.7	0.67	0.7	0.43
0.0	-0.75	0.0	-0.64	0.0	0.24
N = 12, 2 PAIRS		0.0	0.19	0.0	-0.38
		0.7	0.56	0.7	0.79
		0.0	0.18	0.0	0.11
		0.0	0.07	0.0	-0.32
		N = 12, 3 PAIRS		0.7	0.43
				0.0	-0.44
				0.0	0.16
				0.7	0.43
				N = 12, 4 PAIRS	

FIG.2 OUTPUT VALUES
for 12-Neuron GenNets (2,3,4 PAIRS)

FIG.4 shows the output values at the 100th cycle of the elite chromosomes for the two cases. The 5 I/O pair case, shows that for a 16-Neuron GenNet, 5 pairs were too many. Note that for each of these 2 graphs the 4 and 5 I/O pair case GenNets were evolved from scratch, i.e. there was no incremental evolution used.

The chromosome population which resulted from the evolution of the 12-Neuron, 3 I/O pair case, was then "inserted" into the initial population in an experiment to evolve (incrementally) a 16-Neuron, 4 I/O pair case. However these (16-Neuron GenNet) chromosomes needed to be lengthened in order to specify the signs and weights of the connections with the 4 extra neurons. These signs and weights (i.e. slots) of the connections between the original 12 and the extra 4 neurons were concatenated onto the original chromosomes.

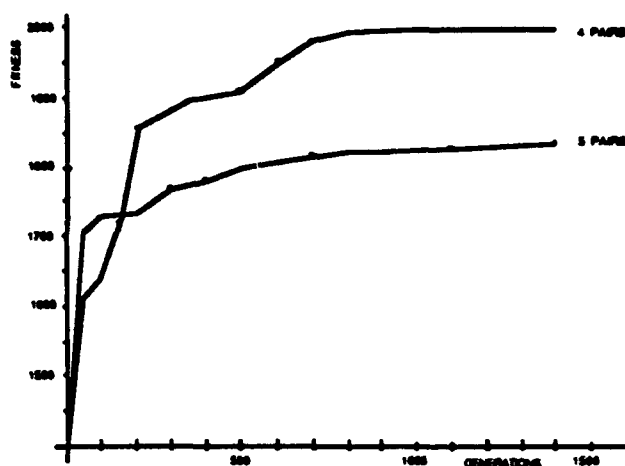


FIG. 3 FITNESS EVOLUTIONS
for 16-Neuron GenNets (4, 5 PAIRS)

TARG	OUTS	TARG	OUTS
0.0	-0.56	0.0	-0.14
0.7	0.72	0.7	0.40
0.7	0.69	0.7	0.49
0.0	-0.63	0.0	-0.03
0.0	-0.14	0.0	0.13
0.0	0.01	0.0	0.15
0.7	0.84	0.7	0.53
0.0	-0.13	0.0	0.29
0.0	-0.49	0.0	-0.48
0.7	0.76	0.7	0.60
0.0	-0.04	0.0	0.10
0.0	-0.52	0.0	-0.62
0.7	0.68	0.7	0.43
0.0	-0.80	0.0	-0.38
0.0	-0.26	0.0	0.33
0.7	0.64	0.7	0.63

N = 16, 4 PAIRS

N = 16, 5 PAIRS

FIG. 4 OUTPUT VALUES
for 16-Neuron GenNets (4, 5 PAIRS)

FIG. 5 shows how this was done. In any region, slots were read from left to right and then from top to bottom. This representation of the chromosome was chosen for two rather obvious reasons. The first reason was that adding further neurons (and hence extending the length of the chromosome) would not change the interpretation of the information

contained in the earlier part of the chromosome (e.g. Region "A" is interpreted in the same way, i.e. codes for the connection weights and signs between the original 12 neurons, even when 4 more neurons are added).

The second reason was that this representation also allows incremental evolution, i.e. one can "load" or insert a smaller (earlier) chromosome (which results from an earlier phase of evolution) into a later, larger chromosome for a second phase of evolution. For example, in FIG. 5 one could load the first 12*12 slots (which resulted from the evolution of a 12-Neuron GenNet) into Region "A" of the 16*16 slots of the chromosome representing a 16-Neuron GenNet.

Thus, to perform the incremental evolution of this experiment, the 12*12 (Region "A") weight matrix which resulted from the evolution of the chromosome of the 12-Neuron GenNet with 3 I/O pairs, was loaded into region "A" of the chromosomes for a 16-Neuron GenNet. Actually an initial population of chromosomes for a 16-Neuron GenNet was randomly generated, followed by the overwriting of the Region "A" of each chromosome of that 16-Neuron GenNet population by the 12-Neuron GenNet chromosome population.

One was then curious to see if the evolution of this "incremented" 16-Neuron GenNet using 4 I/O pairs would occur at all, and if so, would it occur faster than the evolution from scratch of the 16-Neuron GenNet with 4 I/O pairs, as shown in FIG. 3. Intuitively, one feels that since part of the search space of the incremented 16-Neuron GenNet has already been searched, i.e. the "portion" of the search space of the 12-Neuron GenNet that is inserted into the 16-Neuron GenNet chromosomes, then the evolution of the remaining search space which is added with the addition of the 4 new neurons, would be quicker than having to evolve the whole search space corresponding to a 16-Neuron GenNet from scratch.

Whether the final stabilized fitness value of the elite incremented chromosome would be the same as that for the evolution-from-scratch case is difficult to predict. FIG. 6 shows the results. The incremented GenNet obviously

evolved, which is the first and most critical result. Thus a GenNet can be incrementally GPed. Secondly, it evolved more quickly than the "from scratch" case (whose fitness curve is copied over from FIG. 3) for the first 500 generations or so. Thirdly, the fitness value stabilized at a value noticeably lower than the "from scratch" case. This was thought to be significant. It will be interesting to see whether other researchers, when employing incremental evolutionary techniques to other applications, observe the same "better sooner, worse later" phenomenon. If so, then this new "effect" might be worthy of being given a name.

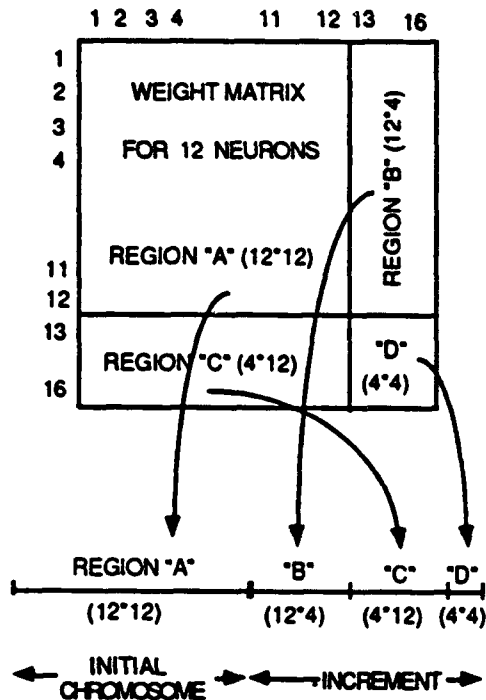


FIG. 5 INCREMENTED WEIGHT MATRIX

When one compares the time taken to evolve a 12-Neuron, 3 I/O pair GenNet ($T_{12,3p}$), plus the time taken to incrementally evolve (from this 12-Neuron GenNet) a 16-Neuron, 3+1 I/O pair GenNet ($T_{12+4,3+1p}$), and compare this sum ($T_{12,3p} + T_{12+4,3+1p}$) with the time taken to evolve a 16-Neuron, 4 I/O pair GenNet from scratch ($T_{16,4p}$), one observes from FIGs. 1 and 6, that approximately :-

$$T_{12,3p} = 200 \text{ generations (fitness > 1900.0),}$$

$$\begin{aligned} T_{12+4,3+1p} &= 350 \text{ generations (fitness > 1900.0),} \\ T_{16,4p} &= 500 \text{ generations (fitness > 1900.0)} \end{aligned}$$

Therefore, $T_{12,3p} + T_{12+4,3+1p}$ and $T_{16,4p}$ are of comparable size (i.e. 550 and 500), i.e. it takes about the same time to perform a two step incremental evolution, as to perform the same size evolution from scratch. BUT, note that $T_{12+4,3+1p}$ is *quicker* than $T_{16,4p}$. These comparative times are debatable, because they depend upon the fitness levels chosen which define when evolutions are "completed".

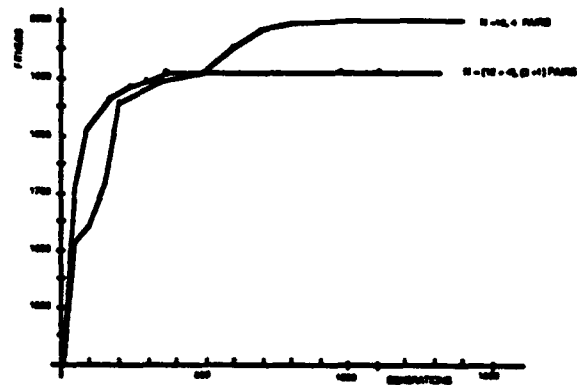


FIG. 6 FITNESS EVOLUTIONS for 16- and (12+4)-Neuron GenNets

TARG	OUTS	TARG	OUTS
0.0	-0.56	0.0	-0.04
0.7	0.72	0.7	0.75
0.7	0.69	0.7	0.46
0.0	-0.63	0.0	-0.54
0.0	-0.14	0.0	-0.20
0.0	0.01	0.0	-0.42
0.7	0.84	0.7	0.47
0.0	-0.13	0.0	-0.53
0.0	-0.49	0.0	-0.03
0.7	0.76	0.7	0.72
0.0	-0.04	0.0	0.38
0.0	-0.52	0.0	-0.40
0.7	0.68	0.7	0.27
0.0	-0.80	0.0	0.29
0.0	-0.26	0.0	-0.46
0.7	0.64	0.7	0.63

N = 16, 4 PAIRS N = 12+4, 3+1 PAIRS

FIG. 7 OUTPUT VALUES for 16-Neuron & (12+4)-Neuron GenNets (4 Prs) & (3+1 Prs)

3. Conclusions and Future Work

One can draw some tentative conclusions from the above work. Firstly, and most critically, in this one case at least, incremental evolution worked (but the final quality seemed to be lower than a from-scratch approach). A lot more work needs to be done by other researchers with other examples to check if these initial results obtained by the author are generally true. As a sideline, it also appeared that the total time taken to perform the initial evolution plus the incremented evolution ($T_{12,3p} + T_{12+4,3+1p}$) was roughly equal to the from-scratch evolution ($T_{16,4p}$). So does this mean that future research teams using GP methods to build their artificial nervous systems have a choice of two approaches, i.e. either to increment or rebuild? The question remains open. However, GIVEN that one already has an evolved GenNet, it is quicker to evolve it incrementally than to start over from scratch (i.e. $T_{12+4,3+1p} < T_{16,4p}$). Hopefully, this result will prove to be general, and will apply to artificial nervous systems.

At the time of writing, the author is attempting to GP artificial nervous systems using cellular automata (CA) as a base. The idea is to grow CA trails (3 cells wide), by sending 4 types of "signals" down the middle of the trail. When the signals hit the end of the trail, four types of action can occur, depending upon the type (i.e. the state or color) of the signal (red = turn trail left, green = turn trail right, brown = extend trail one cell, purple = split trail into a T intersection). The sequence of these signals corresponds to a chromosome in a GA, and hence can be evolved. There is thus a mapping between the sequence and a CA network. When two trails collide, a "synapse" is formed which absorbs oncoming signals.

In the first of a two phase process, the trails are laid down. In the second phase, the CA network uses a second set of CA state transition rules to make the network behave like a neural network, whose fitness at performing some desired task can be measured. To make the CA network behave like a neural network, CA state transition rules need to be defined which make a CA system

function like a neuron (with addition of dendritic signal strengths, and axonal output conversion). Axon signals remain at constant strength, but at a synapse, dendritic signal strengths drop off as a function of the distance from the synapse. Thus the greater the distance between the axon/dendrite synapse, the weaker the signal arriving at the neuron. Hence the distances correspond to the weights in conventional neural network formulations. But the distances can be evolved in the first phase. Hence, using GPed "neurite networks" (a neurite is a baby neuron which grows) based on CAs, it is possible to both grow and evolve neural nets. The author's program based on these ideas is called "CREEPER". If these GPed "neurite nets" prove to be evolvable (i.e. their fitnesses improve over time), then the ideas will be extended from 2 to 3 dimensions, and hopefully later to a hardware implementation, using CAMs (i.e. CA Machines). By having a population of these CAMs, plus local micro processors to measure the fitnesses and to control the GA aspects of the evolution, we have the beginnings of a Darwin Machine design. The Brain Builder Group hopes to evolve large numbers of neural modules (GenNets, i.e. GPed neural nets) and connections between them, using such Darwin Machines. The artificial nervous system which results will be destined to control a robot kitten with about 100 "behaviors". It will take several man years of work to build such a biot (biological robot), so the group will be highly motivated not to have to start from scratch each time we want to increase the biot's capabilities. Hence again, the question raised by this paper "How to GP incrementally", is emphasized.

References

- de Garis H., "Genetic Programming: Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules", Hugo de Garis, in Porter B.W. & Mooney R.J. eds., *Proc. 7th. Int. Conf. on Machine Learning*, pp 132-139, Morgan Kaufmann, 1990.
- de Garis H., "*Genetic Programming : GenNets, Artificial Nervous Systems, Artificial Embryos*", Hugo de Garis, WILEY manuscript, 1993.

V. Special Topics and Applications

A Multistrategy Case-Based and Reinforcement Learning Approach to Self-Improving Reactive Control Systems for Autonomous Robotic Navigation

Ashwin Ram and Juan Carlos Santamaría
 College of Computing
 Georgia Institute of Technology
 Atlanta, Georgia 30332-0280
 {ashwin, carlos}@cc.gatech.edu

Abstract

This paper presents a self-improving reactive control system for autonomous robotic navigation. The navigation module uses a schema-based reactive control system to perform the navigation task. The learning module combines case-based reasoning and reinforcement learning to continuously tune the navigation system through experience. The case-based reasoning component perceives and characterizes the system's environment, retrieves an appropriate case, and uses the recommendations of the case to tune the parameters of the reactive control system. The reinforcement learning component refines the content of the cases based on the current experience. Together, the learning components perform on-line adaptation, resulting in improved performance as the reactive control system tunes itself to the environment, as well as on-line learning, resulting in an improved library of cases that capture environmental regularities necessary to perform on-line adaptation. The system is extensively evaluated through simulation studies using several performance metrics and system configurations.

Keywords: Robot navigation, reactive control, case-based reasoning, reinforcement learning, adaptive control.

1 Introduction

Autonomous robotic navigation is defined as the task of finding a path along which a robot can move safely from a source point to a destination point in an obstacle-ridden terrain, and executing

the actions to carry out the movement in a real or simulated world. Several methods have been proposed for this task, ranging from high-level planning methods to reactive control methods.

High-level planning methods use extensive world knowledge and inferences about the environment they interact with (Fikes, Hart & Nilsson, 1972; Sacerdoti, 1975). Knowledge about available actions and their consequences is used to formulate a detailed plan before the actions are actually executed in the world. Such systems can successfully perform the path-finding required by the navigation task, but only if an accurate and complete representation of the world is available to the system. Considerable high-level knowledge is also needed to learn from planning experiences (e.g., Hammond, 1989a; Minton, 1988; Mostow & Bhatnagar, 1987; Segre, 1988). Such a representation is usually not available in real-world environments, which are complex and dynamic in nature. To build the necessary representations, a fast and accurate perception process is required to reliably map sensory inputs to high-level representations of the world. A second problem with high-level planning is the large amount of processing time required, resulting in significant slowdown and the inability to respond immediately to unexpected situations.

Situated or reactive control methods have been proposed as an alternative to high-level planning methods (Arkin, 1989; Brooks, 1986; Kaelbling, 1986; Payton, 1986). In these methods, no planning is performed; instead, a simple sensory representation of the environment

is used to select the next action that should be performed. Actions are represented as simple behaviors, which can be selected and executed rapidly, often in real-time. These methods can cope with unknown and dynamic environmental configurations, but only those that lie within the scope of predetermined behaviors. Furthermore, such methods cannot modify or improve their behaviors through experience, since they do not have any predictive capability that could account for future consequences of their actions, nor a higher-level formalism in which to represent and reason about the knowledge necessary for such analysis.

We propose a self-improving navigation system that uses reactive control for fast performance, augmented with multistrategy learning methods that allow the system to adapt to novel environments and to learn from its experiences. The system autonomously and progressively constructs representational structures that aid the navigation task by supplying the predictive capability that standard reactive systems lack. The representations are constructed using a hybrid case-based and reinforcement learning method without extensive high-level reasoning. The system is very robust and can perform successfully in (and learn from) novel environments, yet it compares favorably with traditional reactive methods in terms of speed and performance. A further advantage of the method is that the system designers do not need to foresee and represent all the possibilities that might occur since the system develops its own "understanding" of the world and its actions. Through experience, the system is able to adapt to, and perform well in, a wide range of environments without any user intervention or supervisory input. This is a primary characteristic that autonomous agents must have to interact with real-world environments.

This paper is organized as follows. Section 2 presents a technical description of the system, including the schema-based reactive control component, the case-based and reinforcement learning methods, and the system-environment model representations, and places it in the context of related work in the area. Section 3 presents several experiments that evaluate the system. The results shown provide empirical validation of our approach. Section 4 concludes with a discus-

sion of the lessons learned from this research and suggests directions for future research.

2 Technical Details

2.1 System Description

The Self-Improving Navigation System (SINS) consists of a navigation module, which uses schema-based reactive control methods, and an on-line adaptation and learning module, which uses case-based reasoning and reinforcement learning methods. The navigation module is responsible for moving the robot through the environment from the starting location to the desired goal location while avoiding obstacles along the way. The adaptation and learning module has two responsibilities. The adaptation sub-module performs on-line adaptation of the reactive control parameters to get the best performance from the navigation module. The adaptation is based on recommendations from cases that capture and model the interaction of the system with its environment. With such a model, SINS is able to predict future consequences of its actions and act accordingly. The learning sub-module monitors the progress of the system and incrementally modifies the case representations through experience. Figure 1 shows the SINS functional architecture.

The main objective of the learning module is to construct a model of the continuous sensorimotor interaction of the system with its environment, that is, a mapping from sensory inputs to appropriate behavioral (schema) parameters. This model allows the adaptation module to control the behavior of the navigation module by selecting and adapting schema parameters in different environments. To learn a mapping in this context is to discover environment configurations that are relevant to the navigation task and corresponding schema parameters that improve the navigational performance of the system. The learning method is unsupervised and, unlike traditional reinforcement learning methods, does not rely on an external reward function (cf. Watkins, 1989; Whitehead & Ballard, 1990). Instead, the system's "reward" depends on the similarity of the observed mapping in the current environment to the mapping represented in the model. This causes the system to converge towards those mappings that are consistent over

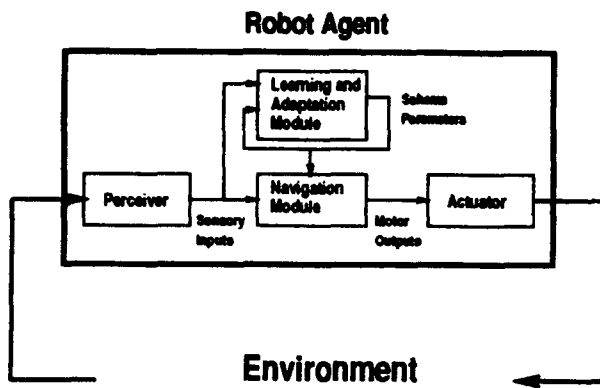


Figure 1: System architecture

a set of experiences.

The representations used by SINS to model its interaction with the environment are initially under-constrained and generic; they contain very little useful information for the navigation task. As the system interacts with the environment, the learning module gradually modifies the content of the representations until they become useful and provide reliable information for adapting the navigation system to the particular environment at hand.

The learning and navigation modules function in an integrated manner. The learning module is always trying to find a better model of the interaction of the system with its environment so that it can tune the navigation module to perform its function better. The navigation module provides feedback to the learning module so it can build a better model of this interaction. The behavior of the system is then the result of an equilibrium point established by the learning module which is trying to refine the model and the environment which is complex and dynamic in nature. This equilibrium may shift and need to be re-established if the environment changes drastically; however, the model is generic enough at any point to be able to deal with a very wide range of environments.

We now present the reactive module, the representations used by the system, and the methods used by the learning module in more detail.

2.2 The Schema-Based Reactive Control Module

The reactive control module is based on the AuRA architecture (Arkin, 1989), and consists

of a set of motor schemas that represent the individual motor behaviors available to the system. Each schema reacts to sensory information from the environment, and produces a velocity vector representing the direction and speed at which the robot is to move given current environmental conditions. The velocity vectors produced by all the schemas are then combined to produce a potential field that directs the actual movement of the robot. Simple behaviors, such as wandering, obstacle avoidance, and goal following, can combine to produce complex emergent behaviors in a particular environment. Different emergent behaviors can be obtained by modifying the simple behaviors. This allows the system to interact successfully in different environmental configurations requiring different navigational "strategies" (Clark, Arkin, & Ram, 1992).

A detailed description of schema-based reactive control methods can be found in Arkin (1989). In this research, we used three motor schemas: AVOID-STATIC-OBSTACLE, MOVE-TO-GOAL, and NOISE. AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles. MOVE-TO-GOAL schema directs the system to move towards a particular point in the terrain. The NOISE schema makes the system to wander in a random direction. Each motor schema has a set of parameters that control the potential field generated by the motor schema. In this research, we used the following parameters: **Obstacle-Gain**, associated with AVOID-STATIC-OBSTACLE, determines the magnitude of the repulsive potential field generated by the obstacles perceived by the system; **Goal-Gain**, associated with MOVE-TO-GOAL, determines the magnitude of the attractive potential field generated by the goal; **Noise-Gain**, associated with NOISE, determines the magnitude of the noise; and **Noise-Persistence**, also associated with NOISE, determines the duration for which a noise value is allowed to persist.

Different combinations of schema parameters produce different behaviors to be exhibited by the system (see figure 2). Traditionally, parameters are fixed and determined ahead of time by the system designer. However, on-line selection and modification of the appropriate parameters based on the current environment can en-

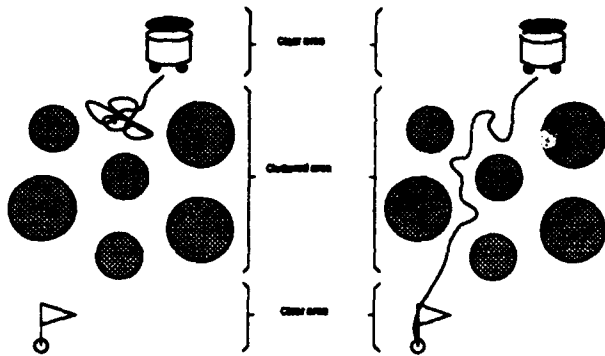


Figure 2: Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing it to "squeeze" through the obstacles and then take a relatively direct path to the goal.

hance navigational performance (Clark, Arkin, & Ram, 1992; Moorman & Ram, 1992). SINS adopts this approach by allowing schema parameters to be modified dynamically. However, in their systems, the cases are supplied by the designer using hand-coded cases. Our system, in contrast, can learn and modify its own cases through experience. The representation of our cases is also considerably different and is designed to support reinforcement learning.

2.3 The System-Environment Model Representation

The navigation module in SINS can be adapted to exhibit many different behaviors. SINS improves its performance by learning how and when to tune the navigation module. In this way, the system can use the appropriate behavior in each environmental configuration encountered. The learning module, therefore, must learn about and discriminate between different environments, and associate with each the appropriate adaptations to be performed on the motor schemas. This requires a representational scheme to model, not just the environment, but the interaction between the system and the environment. However, to ensure that the system does not get bogged down in extensive high-level reasoning, the knowledge represented in the model must be based on perceptual and motor information easily available at the reactive level.

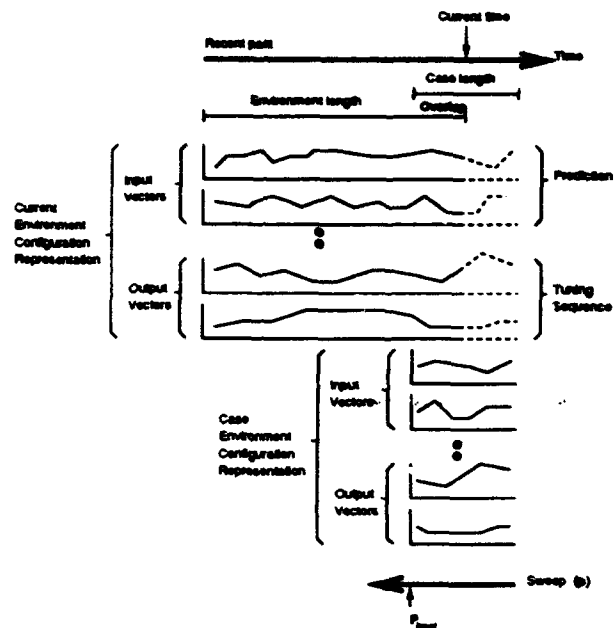


Figure 3: Sample representations showing the time history of analog values representing perceived inputs and schema parameters. Each graph in the case (below) is matched against the corresponding graph in the current environment (above) to determine the best match, after which the remaining part of the case is used to guide navigation (shown as dashed lines).

SINS uses a model consisting of associations between the sensory inputs and schema parameters values. Each set of associations is represented as a case. Sensory inputs provides information about the configuration of the environment, and schema parameter information specifies how to adapt the navigation module in the environments to which the case is applicable. Each type of information is represented as a vector of analog values. Each analog value corresponds to a quantitative variable (a sensory input or a schema parameter) at a specific time. A vector represents the trend or recent history of a variable. A case models an association between sensory inputs and schema parameters by grouping their respective vectors together. Figure 3 show an example of this representation.

This representation has three essential properties. First, the representation is capable of capturing a wide range of possible associations between of sensory inputs and schema parameters. Second, it permits continuous progressive refinement of the associations. Finally, the representation captures trends or patterns of input and

output values over time. This allows the system to detect patterns over larger time windows rather than having to make a decision based only on instantaneous values of perceptual inputs.

In this research, we used four input vectors to characterize the environment and discriminate among different environment configurations: **Obstacle-Density** provides a measure of the occupied areas that impede navigation; **Absolute-Motion** measures the activity of the system; **Relative-Motion** represents the change in motion activity; and **Motion-Towards-Goal** specifies how much progress the system has actually made towards the goal. These input vectors are constantly updated with the information received from the sensors.

We also used four output vectors to represent the schema parameter values used to adapt the navigation module, one for each of the schema parameters (**Obstacle-Gain**, **Goal-Gain**, **Noise-Gain**, and **Noise-Persistence**) discussed earlier. The values are set periodically according to the recommendations of the case that best matches the current environment. The new values remain constant until the next setting period.

The choice of input and output vectors was based on the complexity of their calculation and their relevance to the navigation task. The input vectors were chosen to represent environment configurations in a generic manner but taking into account the processing required to produce those vectors (e.g., obstacle density is more generic than obstacle position, and can be obtained easily from the robot's ultrasonic sensors). The output vectors were chosen to represent directly the actions that the learning module uses to tune the navigation module, that is, the schema parameter values themselves.

2.4 The On-Line Adaptation And Learning Module

This module creates, maintains and applies the case representations used for on-line adaptation of the reactive module. The objective of the learning method is to detect and discriminate among different environment configurations, and to identify the appropriate schema parameter values to be used by the navigation module, in a dynamic and an on-line manner. This means that, as the system is navigating,

the learning module is perceiving the environment, detecting an environment configuration, and modifying the schema parameters of the navigation module accordingly, while simultaneously updating its own cases to reflect the observed results of the system's actions in various situations.

The method is based on a combination of ideas from case-based reasoning and learning, which deals with the issue of using past experiences to deal with and learn from novel situations (e.g., see Kolodner, 1988; Hammond, 1989b), and from reinforcement learning, which deals with the issue of updating the content of system's knowledge based on feedback from the environment (e.g., see Sutton, 1992). However, in traditional case-based planning systems (e.g., Hammond, 1989a) learning and adaptation requires a detailed model of the domain. This is exactly what reactive planning systems are trying to avoid. Earlier attempts to combine reactive control with classical planning systems (e.g., Chien, Gervasio, & DeJong, 1991) or explanation-based learning systems (e.g., Mitchell, 1990) also relied on deep reasoning and were typically too slow for the fast, reflexive behavior required in reactive control systems. Unlike these approaches, our method does not fall back on slow non-reactive techniques for improving reactive control.

To effectively improve the performance of the navigation task, the learning module must find a consistent mapping from environment configurations to control parameters. The learning module captures this mapping in the learned cases, each case representing a portion of the mapping localized in a specific environment configuration. The set of cases represents the system's model of its interactions with the environment, which is adapted through experience using the case-based and reinforcement learning methods. The case-based method selects the case best suited for a particular environment configuration. The reinforcement learning method updates the content of a case to reflect the current experience, such that those aspects of the mapping that are consistent over time tend to be reinforced. Since the navigation module implicitly provides the bias to move to the goal while avoiding obstacles, mappings that are consis-

tently observed are those that tend to produce this behavior. As the system gains experience, therefore, it improves its own performance at the navigation task.

Each case represents an observed regularity between a particular environmental configuration and the effects of different actions, and prescribes the values of the schema parameters that are most appropriate (as far as the system knows based on its previous experience) for that environment. The learning module performs the following tasks in a cyclic manner: (1) **perceive** and represent the current environment; (2) **retrieve** a case whose input vector represents an environment most similar to the current environment; (3) **adapt** the schema parameter values in use by the reactive control module by installing the values recommended by the output vectors of the case; and (4) **learn** new associations and/or adapt existing associations represented in the case to reflect any new information gained through the use of the case in the new situation to enhance the reliability of their predictions.

A detailed description of each step would require more space than is available in this paper; however, a short description of the method follows. The **perceive** step builds a set of four input vectors $E_{input,j}$, one for each sensory input j described earlier, which are matched against the corresponding input vectors $C_{input,j}^k$ of the cases in the system's memory in the **retrieve** step. The case similarity metric SM is based on the mean squared difference between each of the vector values $C_{input,j}^k(i)$ of the k th case C^k over a trending window l_C , and the vector values $E_{input,j}(i)$ of the environment E over a trending window of a given length l_E :

$$SM(E, C^k, p) = \sum_{j=1}^4 w_j \sum_{i=0}^{\min(l_E-p, l_C)} \frac{(E_{input,j}(i+p) - C_{input,j}^k(i))^2}{(\min(l_E-p, l_C) - p)^2}$$

The match window p_{best} is calculated using a reverse sweep over the time axis p similar to a convolution process to find the relative position (represented by $\min(l_E - p, l_C)$) that matches best. The best matching case $C^{k_{best}}$, satisfying the equation:

$$\{k_{best}, p_{best} | \min(SM(E, C^k, p)), \forall k, 0 \leq p \leq l_C\}$$

is handed to the **adapt** step, which selects the schema parameter values $C_{output}^{k_{best}}$ from the output vectors of the case and modifies the values currently in use using a reinforcement formula which uses the case similarity metric as a scalar reward. Thus the actual adaptations performed depend on the goodness of match between the case and the environment, and are given by:

$$C_{output}^{k_{best}} \min(l_E - p_{best}, l_C) \times [1 - RSM] \text{random}(0, \max_k C_{output}^{k_{best}})$$

where RSM is the relative similarity metric discussed below. The random factor allows the system to "explore" the search space locally in order to discover regularities, since the system does not start with prior knowledge that can be used to guide this search.

Finally, the **learn** step uses statistical information about prior applications of the case to determine whether information from the current application of the case should be used to modify this case, or whether a new case should be created. The vectors encoded in the cases are adapted using a reinforcement formula in which a *relative similarity measure* is used as a scalar reward or reinforcement signal. The relative similarity measure RSM , given by $(SM - SM_{best}) / (\overline{SM} - SM_{best})$ quantifies how similar the current environment configuration is to the environment configuration encoded by the case relative to how similar the environment has been in previous utilizations of the case. Intuitively, if case matches the current situation better than previous situations it was used in, it is likely that the situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile modifying the case in the direction of the current situation. Alternatively, if the match is not quite as good, the case should not be modified because that will take it away from the regularity it was converging towards. Finally, if the current situation is a very bad fit to the case, it makes more sense to create a new case to represent what is probably a new class of situations.

Thus, if the RSM is below a certain threshold (0.1 in this paper), the input and output case vectors are updated using a gradient descent formula based on the similarity measure:

$$C_j^{k_{best}}(i) =$$

$$\alpha \min(l_E - p, l_C)(E_j(i + p) - C_j^k(i)), \\ 0 \leq i \leq l_C$$

where the constant α determines the learning rate (0.5 in this paper). In the **adapt** and **learn** steps, the overlap factor $\min(l_E - p_{\text{best}}, l_C)$ is used to attenuate the modification of early values within the case which contribute more to the selection of the current case.

Since the reinforcement formula is based on a relative similarity measure, the overall effect of the learning process is to cause the cases to converge on stable associations between environment configurations and schema parameters. Stable associations represent regularities in the world that have been identified by the system through its experience, and provide the predictive power necessary to navigate in future situations. The assumption behind this method is that the interaction between the system and the environment can be characterized by a finite set of causal patterns or associations between the sensory inputs and the actions performed by the system. The method allows the system to learn these causal patterns and to use them to modify its actions by updating its schema parameters as appropriate.

Genetic algorithms may also be used to modify schema parameters in a given environment (Pearce, Arkin, & Ram, 1992). However, while this approach is useful in the initial design of the navigation system, it cannot change schema parameters during navigation when the system faces environments that are significantly different from the environments used in the training phase of the genetic algorithm. Another approach to self-organizing adaptive control is that of Verschure, Kröse, & Pfeifer (1992), in which a neural network is used to learn how to associate conditional stimulus to unconditional responses. Although their system and ours are both self-improving navigation systems, there is a fundamental difference on how the performance of the navigation task is improved. Their system improves its navigation performance by learning how to incorporate new input data (i.e., conditional stimulus) into an already working navigation system, while SINS improves its navigation performance by learning how to adapt the system itself (i.e., the navigation module). Our system does not rely on new sensory input, but on pat-

terns or regularities detected in perceived environment. Our learning methods are also similar to Sutton (1990), whose system uses a trial-and-error reinforcement learning strategy to develop a world model and to plan optimal routes using the evolving world model. Unlike this system, however, SINS does not need to be trained on the same world many times, nor are the results of its learning specific to a particular world, initial location, or destination location.

3 Evaluation

The methods presented above have been evaluated using extensive simulations across a variety of different types of environment, performance criteria, and system configurations. The objective of these experiments is to measure qualitatively and quantitatively improvement of the navigation performance of SINS (the "adaptive system"), and to compare this performance against a non-learning schema-based reactive system (the "static system") and a system that changes the schema parameter values randomly after every control interval (the "random system"). Rather than simply measure the improvement in performance in SINS by some given metric such as "speedup", we were interested in systematically evaluating the effects of various design decisions on the performance of the system across a variety of metrics in different types of environments. To achieve this, we designed several experiments, which can be grouped into four sets as discussed below.

3.1 Experiment Design

The systems were tested on randomly generated environments consisting of rectangular bounded worlds. Each environment contains circular obstacles, a start location, and a destination location, as shown in figure 2. Figure 4 shows an actual run of the static and adaptive systems on one of the randomly generated worlds. The location, number and radius of the obstacles were randomly determined to create environments of varying amounts of **clutter**, defined as the ratio of free space to occupied space. We tested the effect of three different parameters in the SINS system: **max-cases**, the maximum number of cases that SINS is allowed to create; **case-length**, l_C , representing the time window of a

case; and **control-interval**, which determines how often the schema parameters in the reactive control module are adapted.

We used six estimators to evaluate the navigation performance of the systems. These metrics were computed using a cumulative average over the test worlds to factor out the intrinsic differences in difficulty of different worlds. *Average number of worlds solved* indicates in how many of the worlds posed the system actually found a path to the goal location. The optimum value is 100% since this would indicate that every world presented was successfully solved. *Average steps* indicates the average of number of steps that the robot takes to terminate each world; smaller values indicate better performance. *Average distance* indicates the total distance traveled per world on average; again, smaller values indicate better performance. *Average $\frac{\text{actual}}{\text{optimal}}$ distance per world* indicates the ratio of the total distance traveled and the Euclidean distance between the start and end points, averaged over the solved worlds. The optimal value is 1, but this is only possible in a world without obstacles. *Average virtual collisions per world* indicates the total number of times the robot came within a predefined distance of an obstacle. Finally, *average time per world* indicates the total time the system takes to execute a world on average.

The data for the estimators was obtained after the systems terminated each world. This was to ensure that we were consistently measuring the effect of learning across experiences rather than within a single experience (which is less significant on worlds of this size anyway). The execution is terminated when the navigation system reaches its destination or when the number of steps reaches an upper limit (3000 in the current evaluation). The latter condition guarantees termination since some worlds are unsolvable by one or both systems.

In this paper, we discuss the results from the following sets of experiments:

Experiment set 1: Effect of the multistrategy learning method. We first evaluated the effect of our multistrategy case-based and reinforcement learning method by comparing the performance of the SINS system against the static and random systems. SINS was allowed to learn

up to 10 cases (**max-cases** = 10), each of **case-length** = 4. Adaptation occurred every **control-interval** = 4 steps. Figure 5 shows the results obtained for each estimator over the 200 worlds. Each graph compares the performance on one estimator of each of the three systems, static, random and adaptive, discussed above.

Experiment set 2: Effect of case parameters. This set of experiments evaluated the effect of two parameters of the case-based reasoning component of the multistrategy learning system, that is, **max-cases** and **case-length**. **control-interval** was held constant at 4, while **max-cases** was set to 10, 20, 40 and 80, and **case-length** was set to 4, 6, 10 and 20. All these configurations of SINS, and the static and random systems, were evaluated using all six estimators on 200 randomly generated worlds of 25% and 50% clutter. The results are shown in figures 6 and 7.

Experiment set 3: Effect of control interval. This set of experiments evaluated the effect of the **control-interval** parameter, which determines how often the adaptation and learning module modifies the schema parameters of the reactive control module. **max-cases** and **case-length** were held constant at 10 and 4, respectively, while **control-interval** was set to 4, 8, 12 and 16. All systems were evaluated using all six estimators on 200 randomly generated worlds of 50% clutter. The results are shown in figure 8.

Experiment set 4: Effect of environmental change. This set of experiments was designed to evaluate the effect of changing environmental characteristics, and to evaluate the ability of the systems to adapt to new environments and learn new regularities. With **max-cases** set to 10, 20, 40 and 80, **case-length** set to 4, 6 and 10, and **control-interval** set to 4, we presented the systems with 200 randomly generated worlds of 25% clutter followed by 200 randomly generated worlds of 50% clutter. The results are shown in figure 9.

3.2 Discussion of Experimental Results

The results in figures 5 through 9 show that SINS does indeed perform significantly better than its non-learning counterpart. To obtain a more detailed insight into the nature of the improvement, let us discuss the experimental results in more detail.

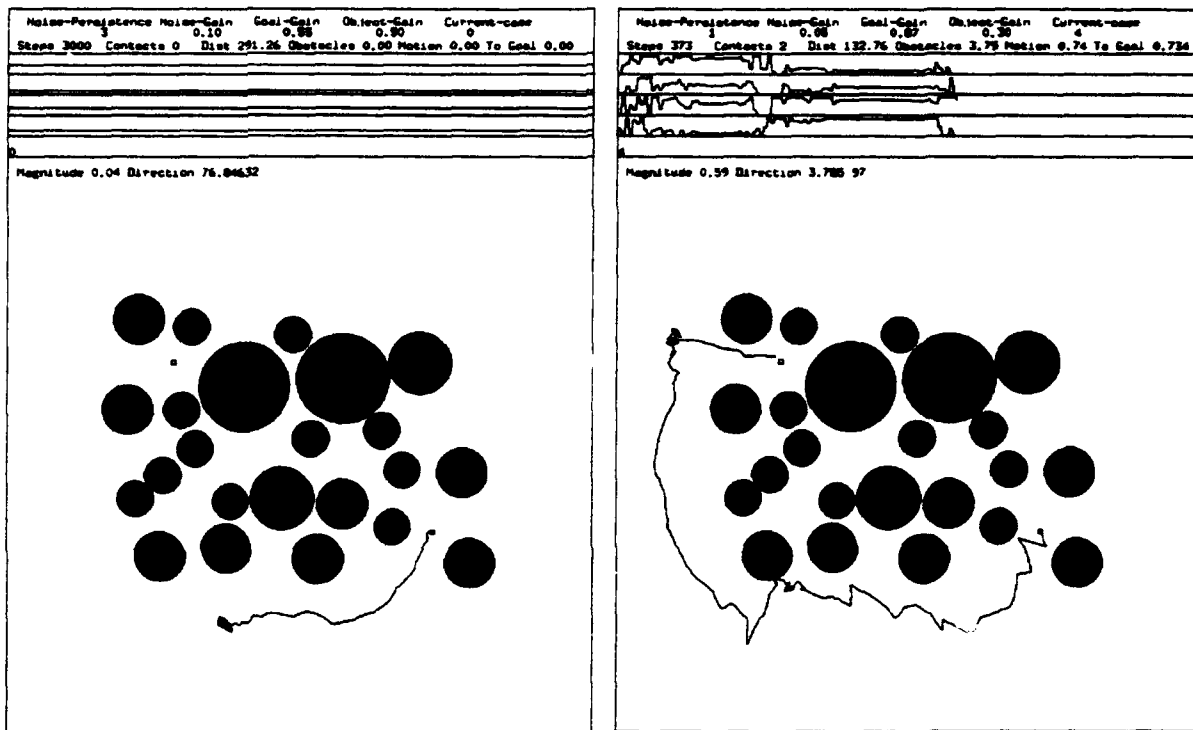


Figure 4: Sample runs of the static and adaptive systems on a randomly generated world. The system starts at the filled box (towards the lower right side of the world) and tries to navigate to the unfilled box. The figure on the left shows the static system. On the right, the adaptive system has learned to "balloon" around the obstacles, temporarily moving away from the goal, and then to "squeeze" through the obstacles (towards the end of the path) and shoot towards the goal. The graphs at the top of the figures plot the values of the schema parameters over the duration of the run.

Experiment set 1: Effect of the multistrategy learning method. Figure 5 shows the results obtained for each estimator over the 200 worlds. As shown in the graphs, SINS performed better than the other systems with respect to five out of the six estimators. Figure 10 shows the final improvement in the system after all the worlds. SINS successfully navigates 93% of the worlds, a 541% improvement over the non-learning system, with 22% fewer virtual collisions. Although the non-learning system was 39% faster, the paths it found required over 4 times as many steps. On average, SINS' solution paths were 25% shorter and required 76% fewer steps, an impressive improvement over a reactive control method which is already good at navigation.

The average time per world was the only estimator in which the self-improving system performed worse. The reason for this behavior is that the case retrieval process is very time consuming. However, since in the physical world the time required for physical execution of a

motor action outweighs the time required to select the action, the time estimator is less critical than the distance, steps, and solved worlds estimators. Furthermore, as discussed below, better case organization methods should reduce the time overhead significantly.

The experiments also demonstrate an somewhat unexpected result: the number of worlds solved by the navigation system is increased by changing the values of the schema parameters even in a random fashion, although the random changes lead to greater distances travelled. This may be due to the fact that random changes can get the system out of "local minima" situations in which the current settings of its parameters are inadequate. However, consistent changes (i.e., those that follow the "regularities" captured by our method) lead to better performance than random changes alone.

Experiment set 2: Effect of case parameters. All configurations of the SINS system navigated successfully in a larger percentage of the test worlds than the static system. Regardless of the

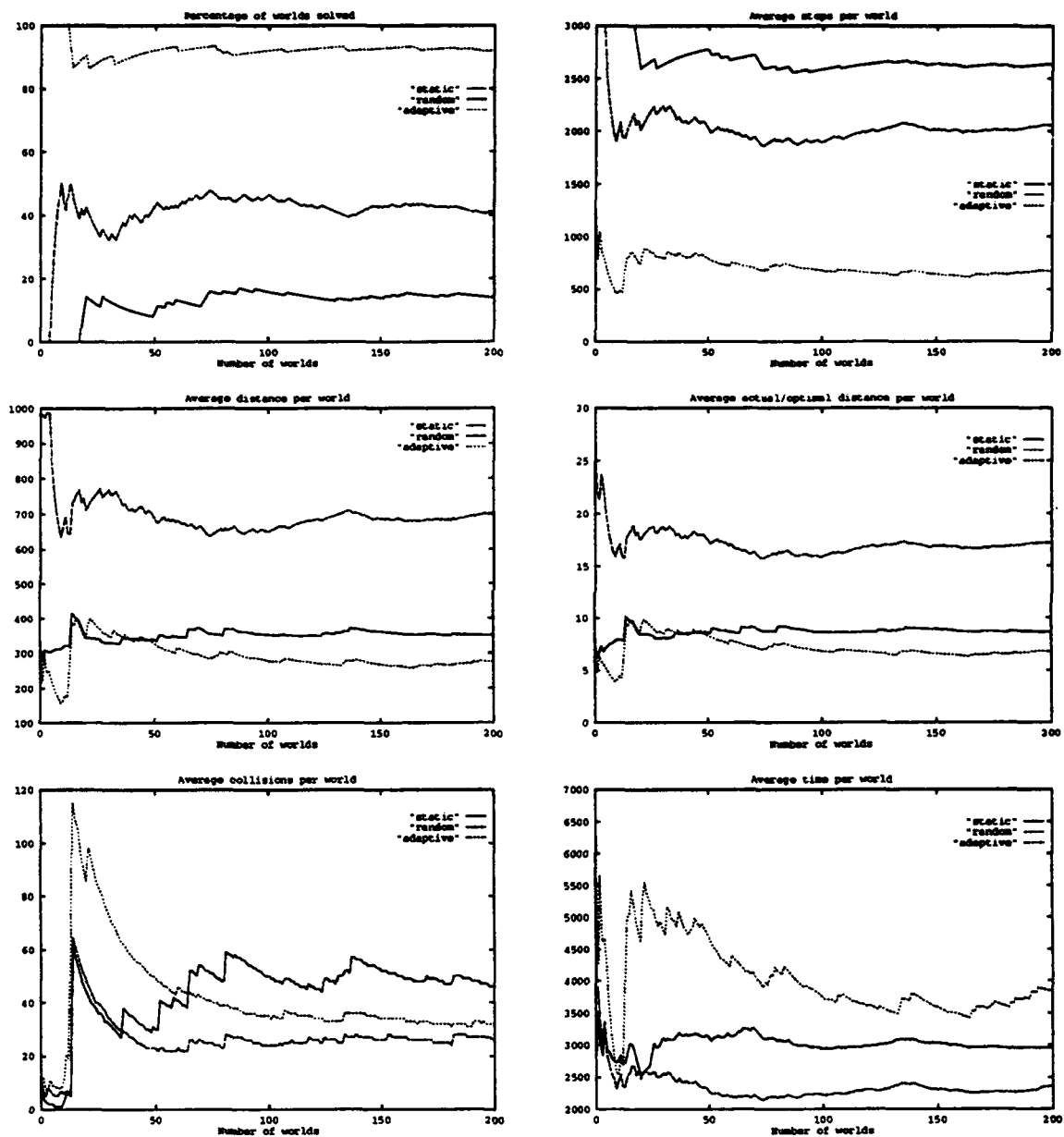
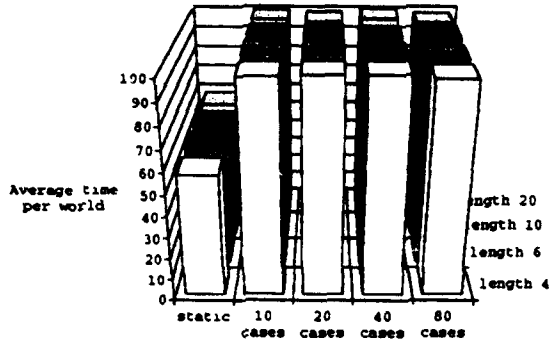
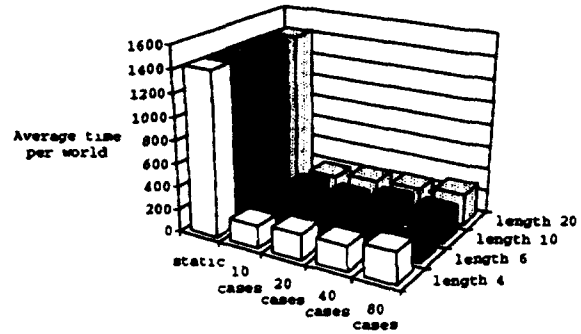


Figure 5: Cumulative performance results.

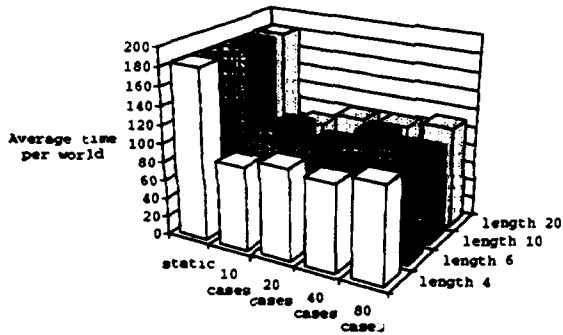
Percentage of worlds solved after 200 experiences



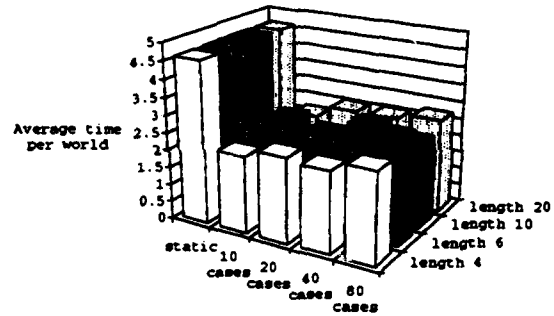
Average steps per world after 200 experiences



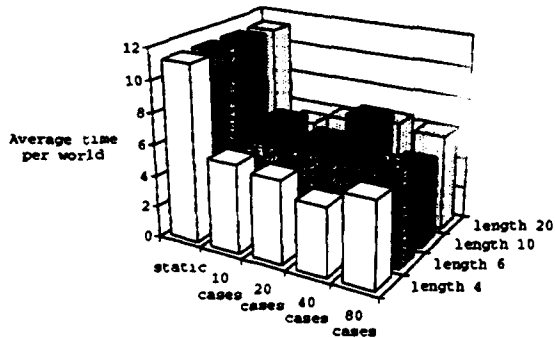
Average distance per world after 200 experiences



Average actual/optimal distance per world after 200 experiences



Average collisions per world after 200 experiences



Average time per world after 200 experiences

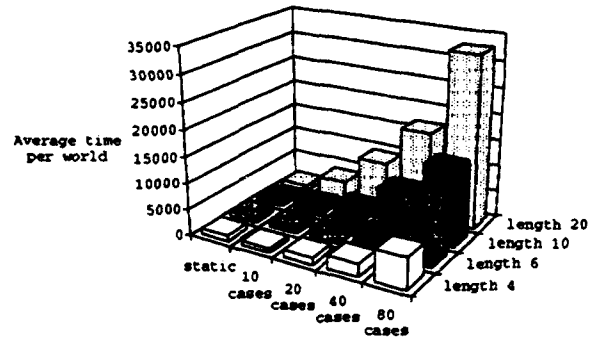
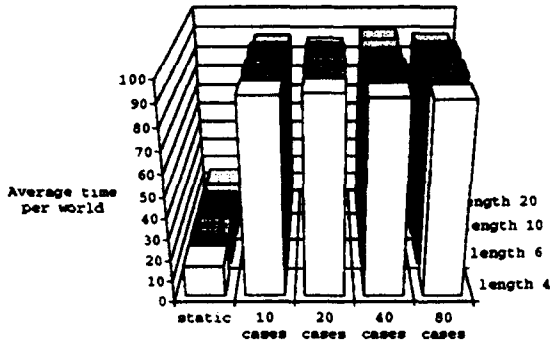
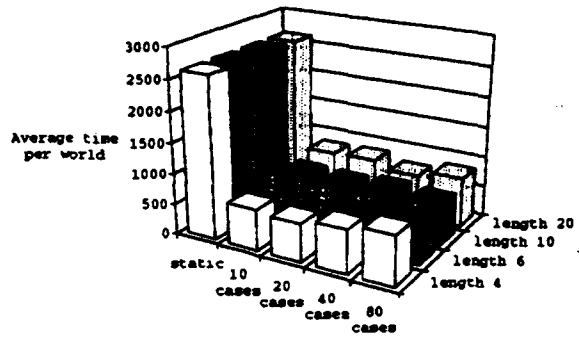


Figure 6: Effect of max-cases and case-length on 25% cluttered worlds.

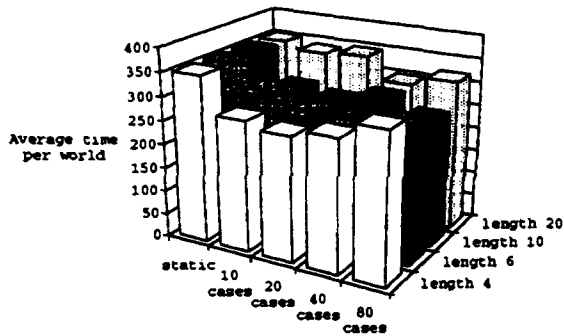
Percentage of worlds solved after 200 experiences



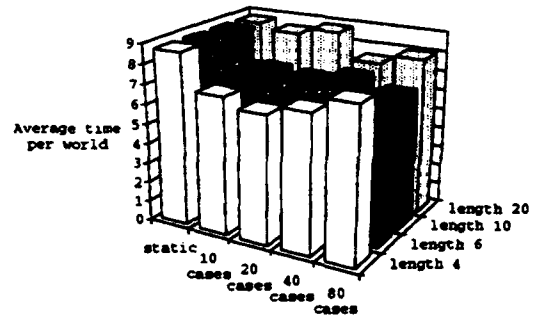
Average steps per world after 200 experiences



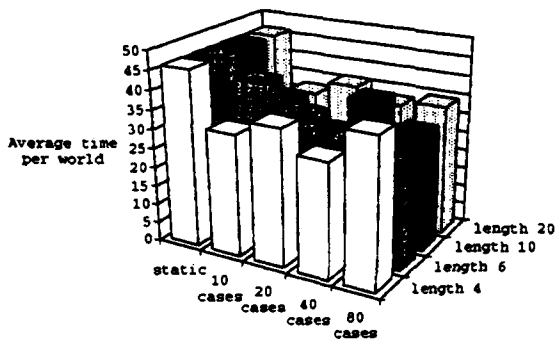
Average distance per world after 200 experiences



Average actual/optimal distance per world after 200 experiences



Average collisions per world after 200 experiences



Average time per world after 200 experiences

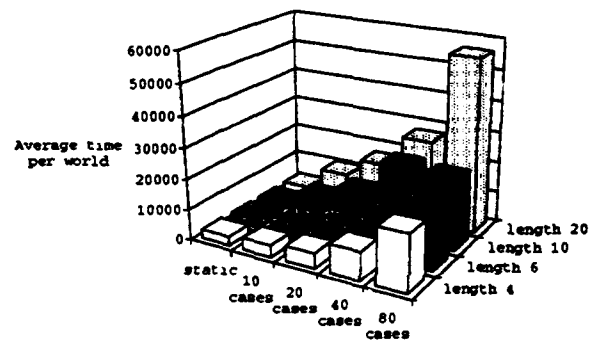


Figure 7: Effect of max-cases and case-length on 50% cluttered worlds.

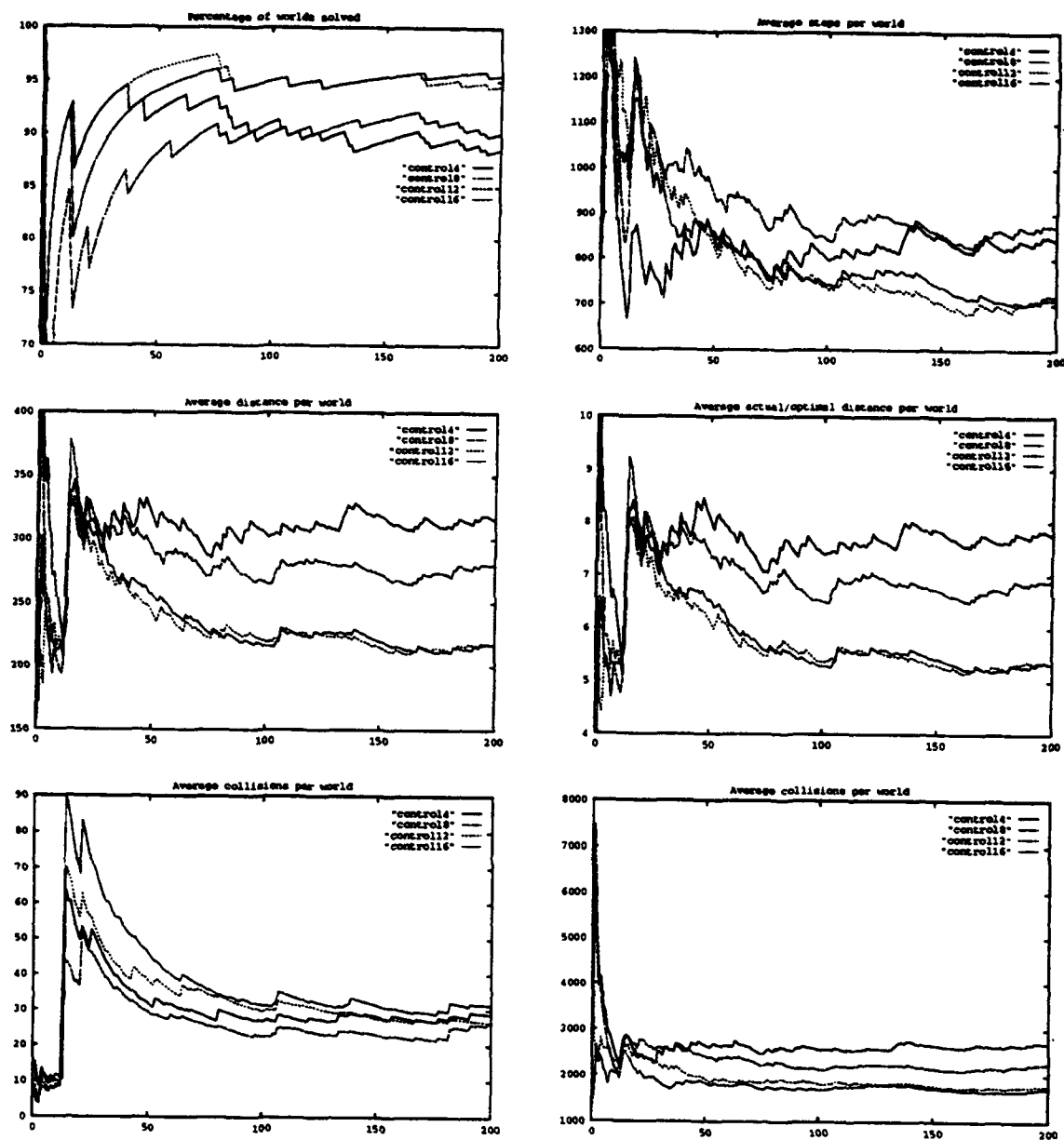


Figure 8: Effect of control-interval.

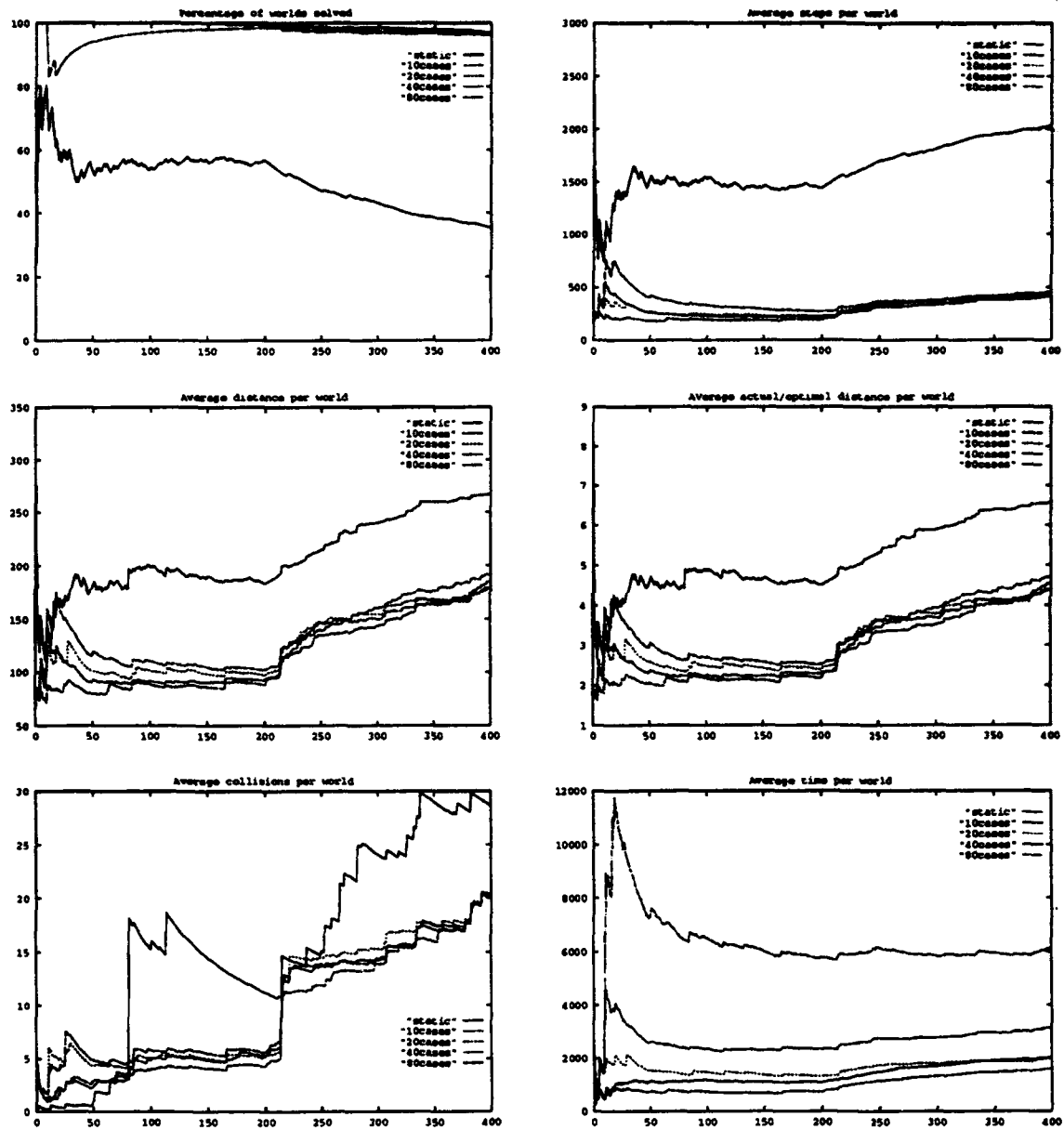


Figure 9: Effect of a sudden change in environment (after the 200th world).

	<i>static</i>	<i>random</i>	<i>adaptive</i>
Percentage of worlds solved	14.5%	41.5%	93%
Average steps per world	2624.6	2046.8	618.4
Average distance per world	350.5	696.5	261.2
Average $\frac{\text{actual}}{\text{optimal}}$ distance	8.6	17.1	6.4
Average virtual collisions	46.1	26.4	35.7
Average time per world, ms	2947.8	2352.5	4878.3

Figure 10: Final performance results.

max-cases and **case-length** parameters, SINS could solve most of the 25% cluttered worlds (as compared with 55% in the static system) and about 90% of the 50% cluttered worlds (as compared with 15% in the static system). Although it could be argued that an alternative set of schema parameters might lead to better performance in the static system, SINS would also start out with those same settings and improve even further upon its initial performance.

Our experiments revealed that, in both 25% and 50% cluttered worlds, SINS needed about 40 worlds to learn enough to be able to perform successfully thereafter using 10 or 20 cases. However, with higher numbers of cases (40 and 80), it took more trials to learn the regularities in the environment. It appears that larger numbers of cases require more trials to train through trial-and-error reinforcement learning methods, and furthermore there is no appreciable improvement in later performance. The **case-length** parameter did not have an appreciable effect on performance in the long run, except on the *average number of virtual collisions* estimator which showed the best results with case lengths of 4 and 10.

As observed earlier in experiment set 1, SINS requires a time overhead for case-based reasoning and thus loses out on the *average time* estimator. Due to the nature of our current case retrieval algorithm, the time required increases linearly with **max-cases** and with **case-length**. In 25% cluttered worlds, values of 10 and 4, respectively, for these parameters provide comparable performance.

Experiment set 3: Effect of control interval. Although all settings resulted in improved performance through experience, the best and worst performance in terms of *average number*

of worlds solved was obtained with **control-interval** set to 12 and 4, respectively. For low **control-interval** values, we expect poorer performance because environment classification cannot occur reliably. We also expect poorer performance for very high values because the system cannot adapt its schema parameters quickly enough to respond to changes in the environment. Other performance estimators also show that **control-interval** = 12 is a good setting. Larger **control-intervals** require less case retrievals and thus improve **average time**; however, this gets compensated by poorer performance on other estimators.

Experiment set 4: Effect of environmental change. The results from these experiments demonstrate the flexibility and adaptiveness of the learning methods used in SINS. Regardless of parameter settings, SINS continued to be able to navigate successfully despite a sudden change in environmental clutter. It continued to solve about 95% of the worlds presented to it, with only modest deterioration in steps, distance, virtual collisions and time in more cluttered environments. The performance of the static system, in contrast, deteriorated in the more cluttered environment.

Summary: These and other experiments show the efficacy of the multistrategy adaptation and learning methods used in SINS across a wide range of qualitative metrics, such as flexibility of the system, and quantitative metrics that measure performance. The results also indicate that a good configuration for practical applications is **max-cases** = 10, **case-length** = 4, and **control-interval** = 12, although other settings might be chosen to optimize particular performance estimators of interest. These values have been determined empirically. Although the empirical

results can be explained intuitively, more theoretical research is needed to analyze why these particular values worked best.

4 Conclusions

We have presented a novel method for augmenting the performance of a reactive control system that combines case-based reasoning for on-line parameter adaptation and reinforcement learning for on-line case learning and adaptation. The method is fully implemented in the SINS program, which has been evaluated through extensive simulations.

The power of the method derives from its ability to capture common environmental configurations, and regularities in the interaction between the environment and the system, through an on-line, adaptive process. The method adds considerably to the performance and flexibility of the underlying reactive control system because it allows the system to select and utilize different behaviors (i.e., different sets of schema parameter values) as appropriate for the particular situation at hand. SINS can be characterized as performing a kind of constructive representational change in which it constructs higher-level representations (cases) from low-level sensorimotor representations (Ram, 1993).

In SINS, the perception-action task and the adaptation-learning task are integrated in a tightly knit cycle, similar to the "anytime learning" approach of Grefenstette & Ramsey (1992). Perception and action are required so that the system can explore its environment and detect regularities; they also, of course, form the basis of the underlying performance task, that of navigation. Adaptation and learning are required to generalize these regularities and provide predictive suggestions based on prior experience. Both tasks occur simultaneously, progressively improving the performance of the system while allowing it to carry out its performance task without needing to "stop and think."

In contrast to traditional case-based reasoning methods which perform high-level reasoning in discrete, symbolic problem domains, SINS is based on a new method for "continuous case-based reasoning" in problem domains that involve continuous information, such as sensori-

motor information for robot navigation (Ram & Santamaría, 1993). There are still several unresolved issues in this research. The case retrieval process is very expensive and limits the number of cases that the system can handle without deteriorating the overall navigational performance, leading to a kind of utility problem (Minton, 1988). Our current solution to this problem is to place an upper bound on the number of cases allowed in the system. A better solution would be to develop a method for organization of cases in memory; however, conventional memory organization schemes used in case-based reasoning systems (see Kolodner, 1992) assume structured, nominal information rather than continuous, time-varying, analog information of the kind used in our cases.

Another open issue is that of the nature of the regularities captured in the system's cases. While SINS' cases do enhance its performance, they are not easy to interpret. Interpretation is desirable, not only for the purpose of obtaining of a deeper understanding of the methods, but also for possible integration of higher-level reasoning and learning methods into the system.

Despite these limitations, SINS is a complete and autonomous self-improving navigation system, which can interact with its environment without user input and without any pre-programmed "domain knowledge" other than that implicit in its reactive control schemas. As it performs its task, it builds a library of experiences that help it enhance its performance. Since the system is always learning, it can cope with major environmental changes as well as fine tune its navigation module in static and specific environment situations.

References

- Arkin, R.C., Motor Schema-Based Mobile Robot Navigation, *The International Journal of Robotics Research*, 8(4):92-112, 1989.
- Brooks, R., A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, RA-2(1):14-23, 1986.
- Chien, S.A., Gervasio, M.T., & DeJong, G.F., On Becoming Decreasingly Reactive: Learning to Deliberate Minimally, in Birnbaum, L. & Collins, G. (editors), *Proceedings of the Eighth*

International Workshop on Machine Learning, 288–292, Chicago, IL, 1991.

Clark, R.J., Arkin, R.C., & Ram, A., Learning Momentum: On-Line Performance Enhancement for Reactive Systems, in *Proceedings of the IEEE International Conference on Robotics and Automation*, 111–116, Nice, France, 1992.

Fikes, R.E., Hart, P.E., & Nilsson, N.J., Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3:251–288, 1972.

Grefenstette, J.J. & Ramsey, C.L. An Approach to Anytime Learning, in Sleeman, D. & Edwards, P. (editors), *Machine Learning: Proceedings of the Ninth International Conference*, 189–195, Aberdeen, Scotland, 1992.

Hammond, K.J. *Case-Based Planning: Viewing Planning as a Memory Task*, Academic Press, Boston, MA, 1989a.

Hammond, K.J. (editor), *Proceedings of the Second Case-Based Reasoning Workshop*, Pensacola Beach, FL, Morgan Kaufman, 1989b.

Kaelbling, L., An Architecture for Intelligent Reactive Systems, Technical Note 400, SRI International, 1986.

Kolodner, J.L. (editor), *Proceedings of a Workshop on Case-Based Reasoning*, Clearwater Beach, FL, Morgan Kaufman, 1988.

Kolodner, J.L., *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, CA, 1992 (in press).

Minton, S., *Learning Effective Search Control Knowledge: An Explanation-Based Approach*, PhD thesis, Technical Report CMU-CS-88-133, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, 1988.

Mitchell, T.M., Becoming Increasingly Reactive, in *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1051–1058, Boston, MA, 1990.

Moorman, K. & Ram, A., A Case-Based Approach to Reactive Control for Autonomous Robots, in *Proceedings of the AAAI Fall Symposium on AI for Real-World Autonomous Mobile Robots*, Cambridge, MA, 1992.

Mostow, J. & Bhatnagar, N., FAILSAFE — A Floor Planner that uses EBG to Learn from its Failures, in *Proceedings of the Tenth Inter-*

national Joint Conference on Artificial Intelligence, 249–255, Milan, Italy, 1987.

Payton, D., An Architecture for Reflexive Autonomous Vehicle Control, in *Proceedings of the IEEE Conference on Robotics and Automation*, 1838–1845, 1986.

Pearce, M., Arkin, R., & Ram, A., The Learning of Reactive Control Parameters through Genetic Algorithms, In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 130–137, Raleigh, NC, 1992.

Ram, A., Creative Conceptual Change, in *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO, 1993 (to appear).

Ram, A. & Santamaría, J.C., Continuous Case-Based Reasoning, in Leake, D.B. (editor), *Proceedings of the AAAI Workshop on Case-Based Reasoning*, Washington, DC, 1993 (to appear).

Sacerdoti, E.D., A Structure for Plans and Behavior, Technical Note 109, Stanford Research Institute, Artificial Intelligence Center. Summarized in P.R. Cohen & E.A. Feigenbaum's *Handbook of AI*, Volume III, pages 541–550, 1975.

Segre, A.M., *Machine Learning of Robot Assembly Plans*, Kluwer Academic Publishers, Norwell, MA, 1988.

Sutton, R.S., Integrated Architectures for Learning, Planning, and Reacting based on Approximating Dynamic Programming, in *Proceedings of the Seventh International Conference on Machine Learning*, 216–224, Austin, TX, 1990.

Sutton, R.S. (editor), *Machine Learning*, 8(3/4), special issue on Reinforcement Learning, 1992.

Verschure, P.F.M.J., Kröse, B.J.A., & Pfeifer, R., Distributed Adaptive Control: The Self-Organization of Structured Behavior. *Robotics and Autonomous Systems*, 9:181–196, 1992.

Watkins, C.J.C.H., *Learning from Delayed Rewards*, PhD thesis, University of Cambridge, England, 1989.

Whitehead, S.D. & Ballard, D.H., Active Perception and Reinforcement Learning, in *Proceedings of the Seventh International Conference on Machine Learning*, 179–188, Austin, TX, 1990.

A Machine Learning Approach To Document Understanding

F. Esposito, D. Malerba, G. Semeraro

Dipartimento di Informatica
Università degli Studi
70126 Bari, Italy

M. Pazzani

Dept. of Information and Computer Science
University of California
92717 Irvine, CA

Abstract

Document understanding denotes the process of identification of logical components of a document and the subsequent extraction of relationships between logical components. In this paper the possibility of *learning recognition* rules for the identification of logical components in a page layout is investigated. For this purpose, FOCL, a system that learns function-free Horn clauses, has been employed and some problems concerning both infinite recursion and the convergence of the learning process have been discussed. Finally, a critic to the underlying independence assumption made by almost all systems that learn from examples is presented and the problem of contextual learning is defined. Some preliminary experimental results show that the definition of a dependence hierarchy between concepts can improve predictive accuracy and decrease learning time for labelling problems like document understanding.

Key words: document understanding, learning dependent concepts, contextual learning

1. Introduction

The automatic classification and understanding of multimedia documents are the fundamental tasks of an intelligent system for office automation, which aims at automatically storing, retrieving and interchanging multimedia office documents. The system is currently developed as a task of the workpackage AP (APplication for automatic classification of documents) of the INTREPID¹ (INnovative Techniques for REcognition and ProcessIng of Documents) project. According to the ODA/ODIF standard (Horak, 1985), any document is characterized by two different structures representing both its content and its internal organization: the *layout* (or *geometric*) structure and the *logical* structure. The former associates the content of the document with a hierarchy of *layout objects* such as text lines, vertical/horizontal lines, graphic elements, photographic elements, columns, pages and so on (Figure 1). The latter associates the content of the document with a hierarchy of *logical objects* such as title, abstract, paragraphs, sections, chapters, tables, figures, and so on (Figure 2).

¹ The work in the INTREPID project is done within the framework of the ESPRIT programme and partly funded by the Commission of the European Communities. The following companies form the consortium: AEG Electrocom (D), CTA (E), Nottingham Polytechnic (GB), Olivetti Systems & Networks (I), Pacer Systems Ltd. (GB), University of Bari (I), University of Koblenz (D) and University of Naples (I).

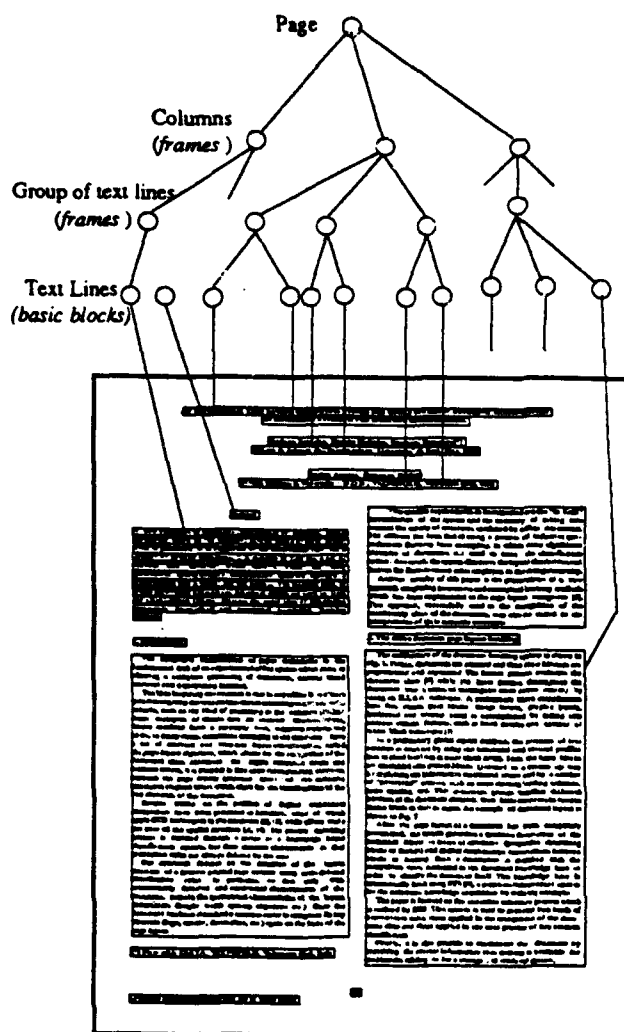


Figure 1. The hierarchical layout structure of a document

Each layout/logical object can be described by a set of attributes. For instance, layout objects can be characterized by the type of content (text, graphics, etc.), their position in the page, their shape, their dimension as well as the numerical properties of their bitmaps, while logical attributes can be described by their type (abstract, paragraph, etc.), some key-words contained in the text (date, figure, etc.), their position.

Relationships among different objects are also possible. Of course, the hierarchy in the layout/logical structures defines some hierarchical relationships among objects of the same structure. However, other, and perhaps more interesting, relationships exist among logical objects (*logical-logical* relationships) and

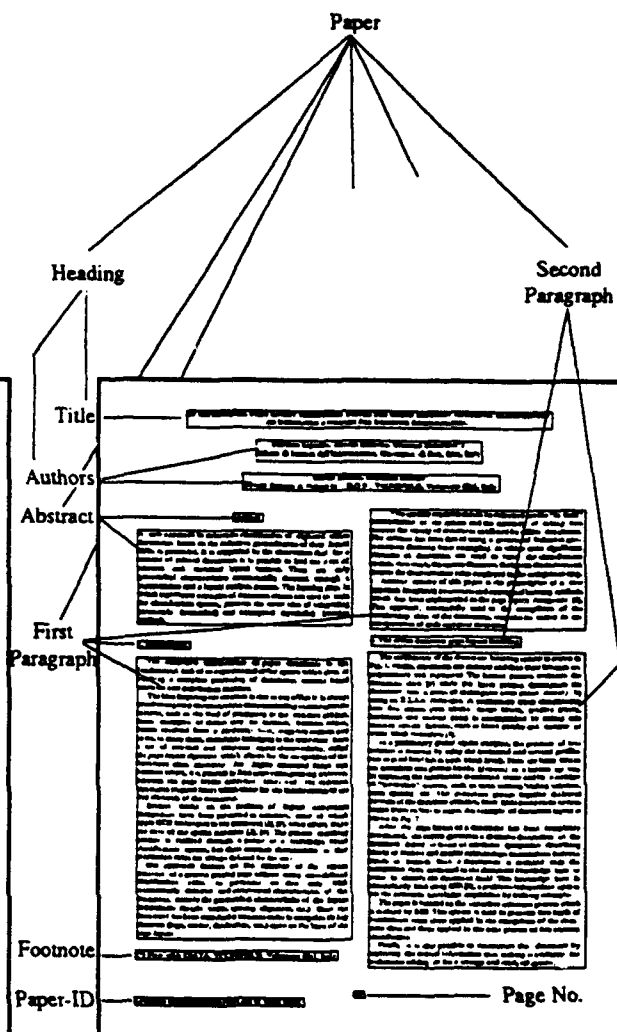


Figure 2. The hierarchical logical structure of a document

among layout objects (*layout-layout* relationships). An example of layout-layout relationship is the mutual position of two layout objects while the cross-reference of a caption to a figure or the reading order of some parts of a document are two examples of logical-logical relationships. Finally, *logical-layout* relationships between one or more elements of the layout hierarchy and one element of the logical hierarchy can be defined. These last are the most interesting, since they allow us to identify some logical components of a document without reading its content by means of an optical character recognizer (OCR) but using only layout (or geometrical) characteristics. For instance, in a standard English letter, the date is

under the sender's address which is in turn in the top left hand corner. Thus, this simple layout information can be profitably exploited by a document management system to identify specific portion of content.

By *document analysis* it is generally meant the process of breaking down a document image into several blocks, which represent layout components, without any knowledge regarding the specific format (Tsujimoto and Asada, 1990).

On the contrary, the term *document understanding* denotes the process of identification of logical components of a document and the subsequent extraction of logical-logical relationships such as the reading order (Tang *et al.*, 1991). When there exist logical-layout relationships due to a standard format of the document, then it is possible to understand a document by using only layout information extracted from the layout analysis process. Furthermore, the identification of text and picture regions is also important in order to limit the application of the OCR, so that only information useful for storing and retrieval purposes is read. Therefore, document analysis always precedes the document understanding phase. Nevertheless, when several kinds of documents have to be automatically handled, document understanding becomes a difficult process due to the different logical-layout relationships met in each kind of document. For instance, letters from various companies will present different writing standards, so the identification of the sender or receiver could be hard if only layout information were used. Thus, an intermediate step becomes necessary: *document classification*, that is the identification of the particular class the document belongs to. Once again, layout information could help to recognize the class of a document when there exists a definite set of relevant and invariant layout characteristics, the so-called *page layout signature*. In (Esposito *et al.*, 1990) a solution to the problem of document classification has been presented. More precisely, the classification rules

for each class of documents can be automatically generated by means of an inductive learning process given a set of significant examples of documents for each class (*training set*). The learning system, named RES, integrates a parametric classifier, in particular Fisher's linear discriminant functions, with a symbolic learning method based on the STAR methodology (Michalski, 1980). The main advantage of adopting a machine learning approach for the problem of document classification is a greater flexibility of the office documents management system since it can be customized more quickly and easily. The success of this approach to document classification induced us to investigate the possibility of adopting the same approach for the problem of document understanding, that is recognizing logical components of a document.

Given a set of documents whose page layouts have already been analyzed and assumed that the user-trainer has already labelled some layout components according to their meaning (e.g., sender or receiver of a letter), the problem is that of learning some rules that allow the correct labelling of layout components to be performed. As said above, the problem of document understanding can be strongly simplified when the class of documents has already been identified. Indeed, in this case we can more easily define logical components for each class of documents and we significantly reduce the variability of training instances for this new learning problem. In spite of such a shrewdness, the problem of learning rules for document understanding is still more complex than the problem of learning recognition rules for classifying documents. In fact, in document understanding, concepts to learn refer to a part of a document rather than to the whole document, and since parts of documents may be related to each other according to logical-logical relationships, this leads to the problem of *learning mutually dependent concepts* (or *contextual rules*). Most of the studies on supervised inductive learning presented in the machine learning

literature make the implicit assumption that concepts are independent (*independence assumption*), and consequently, that training instances are independent. Of course, traditional learning algorithms making the independence assumption can still be exploited for the problem of document understanding by simply neglecting logical-logical relationships, but it is our opinion this is not the correct way to solve the problem. Indeed, document understanding is a particular case of labelling problems, in which the correct label can often be assigned to a part of a complex object only by taking into account spatial relationships with other parts whose labels are already known. Thus, we believe that by taking into account concept dependencies it is possible to generate more accurate and simpler rules, since the learning paradigm is a better approximation of reality. When concept dependencies are intrinsically acyclic, the structure of concept dependencies can be represented by means of a directed acyclic graph. Such a graph can be either provided by the teacher or it can be inferred by means of statistical techniques. The former solution realizes a combination of interactive concept learning and supervised inductive learning, while the latter provides a multistrategy learning methodology that integrates numerical and symbolic learning.

In the next section, a representation language used to describe a page layout will be presented and the opportunity of introducing some intensionally defined predicates in the background knowledge will be discussed. Section 3 is devoted to the problem of learning Horn clauses by means of a well-known learning system: FOCL (Pazzani and Kibler, 1992). Some experimental results on the problem of learning rules for document understanding by means of the traditional strategy are presented in section 4, while the problem of learning dependencies between concepts together with results of the contextual learning strategy for document understanding are shown in section 5.

2. A Representation Language For Page Layout Description

In the general inductive problem (Muggleton, 1992), we are provided with:

- L_O : the language of observations
- L_B : the language of background knowledge
- L_H : the language of hypotheses
- a set of examples or observations, O , described by using L_O
- some background knowledge, B , described by means of L_B

and we want to find a hypothesis H , described in the language L_H , such that:

$$B \wedge H \vdash O$$

Therefore, before describing how hypotheses, i.e. rules for document understanding, are generated, it is necessary to introduce the three languages, L_O , L_B , and L_H .

In the application of document understanding, L_O is the language used to describe instances of different logical objects. Each instance is represented as a ground Horn clause in which different constants represent different layout objects of one or more documents as well as the documents themselves. In particular, a subset of the Horn clauses is used, namely linked Horn clauses (Helft, 1987), since it allows only meaningful hypotheses to be represented. The set of extensionally defined predicates is reported in Table I. A predicate is *extensionally* defined when a list of tuples for which the predicate is true is provided. Figure 3 shows an example of page layout in which some blocks have been labelled. There are five different logical objects, namely sender, receiver, logotype, reference number and date. Other blocks are purposely unlabelled since we are not interested to recognize each part of the document. Obviously, each document is a source of more than one instance of logical components. Therefore, we can write down as many ground Horn clauses as the number of layout objects in a page layout. In Figure 4 the description of the block sender of the document is provided.

Table I
Predicates for the Page Layout Description

Predicate	Meaning
logic_type-sender(X) logic_type-receiver(X) logic_type-logo(X) logic_type-ref(X) logic_type-date(X) logic_type-unsigned(X)	logical label of the layout object X
width-very-very-small(X) width-very-small(X) width-small(X) width-medium-small(X) width-medium(X) width-medium-large(X) width-large(X) width-very-large(X) width-very-very-large(X)	width of the layout object X
height-smallest(X) height-very-very-small(X) height-very-small(X) height-small(X) height-medium-small(X) height-medium(X) height-medium-large(X) height-large(X) height-very-large(X) height-very-very-large(X) height-largest(X)	height of the layout object X
type-text(X) type-hor-line(X) type-picture(X) type-ver-line(X) type-graphic(X) type-mixture(X)	type of the layout object X
part_of(X, Y)	layout object Y belongs to document X
position-top-left(X) position-top(X) position-top-right(X) position-left(X) position-center(X) position-right(X) position-bottom-left(X) position-bottom(X) position-bottom-right(X)	position of the layout object X
on_top(X, Y)	layout object X is on top of layout object Y
to_right(X, Y)	layout object X is to the right of layout object Y
aligned-only-left-col(X, Y) aligned-only-right-col(X, Y) aligned-only-middle-col(X, Y) aligned-both-columns(X, Y) aligned-only-upper-row(X, Y) aligned-only-lower-row(X, Y) aligned-only-middle-row(X, Y) aligned-both-rows(X, Y)	layout objects X and Y are aligned

The language of hypotheses, L_H , generated by FOCL, is a subset of pure Prolog, in which neither functions nor constants occur: Horn clauses of L_H are called function-free. As FOIL (Quinlan, 1990), FOCL adopts Prolog's negation-as-failure rule (Clark, 1978) to define the meaning of a negated predicate. A hypothesis is expressed as a collection of function-free Horn clauses having the same head. Such a collection is called *rule* or *predicate definition*. FOCL allows predicates to be defined intensionally as well, that is it provides a way to introduce some inference rules as background knowledge to use during the induction process.

For the application of document understanding, we defined several inference rules concerning the position of a block, the type of alignment between blocks and the mutual position of blocks. As to the position of blocks, a page was originally split in nine areas by discretizing the numerical coordinates of the centre of each block. However, some logical components may be in different positions but in the same *band* (a band is a set of three contiguous positions in the page). In this case, information on the band may be more useful than the detailed information on the area, since it makes the generation of a rule easier. Therefore, we introduced the following inference rules as background knowledge:

```

top-horiz-band(X) ← position-top-left(X)
top-horiz-band(X) ← position-top(X)
top-horiz-band(X) ← position-top-right(X)
central-horiz-band(X) ← position-left(X)
central-horiz-band(X) ← position-center(X)
central-horiz-band(X) ← position-right(X)
bottom-horiz-band(X) ← position-bottom-left(X)
bottom-horiz-band(X) ← position-bottom(X)
bottom-horiz-band(X) ← position-bottom-right(X)
left-vert-band(X) ← position-top-left(X)
left-vert-band(X) ← position-left(X)
left-vert-band(X) ← position-bottom-left(X)
central-vert-band(X) ← position-top(X)
central-vert-band(X) ← position-center(X)
central-vert-band(X) ← position-bottom(X)
right-vert-band(X) ← position-top-right(X)
right-vert-band(X) ← position-right(X)
right-vert-band(X) ← position-bottom-right(X)

```

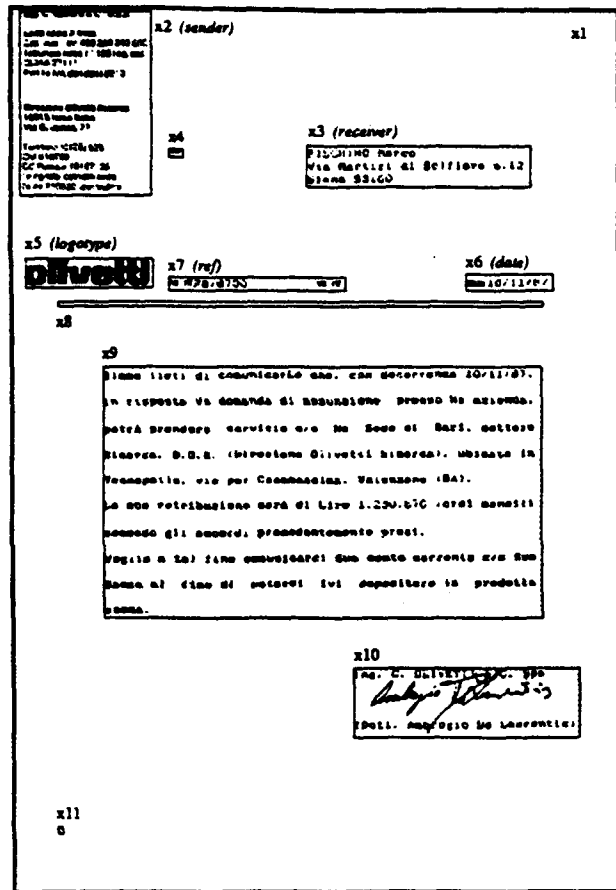


Figure 3. Page layout of a document with labelled blocks.

We also defined the predicates *aligned-by-column* and *aligned-by-row* as follows:

$\text{aligned-by-column}(X,Y) \leftarrow \text{aligned-only-left-col}(X,Y)$
 $\text{aligned-by-column}(X,Y) \leftarrow$

$\text{aligned-only-middle-col}(X,Y)$
 $\text{aligned-by-column}(X,Y) \leftarrow \text{aligned-only-right-col}(X,Y)$
 $\text{aligned-by-column}(X,Y) \leftarrow \text{aligned-both-columns}(X,Y)$
 $\text{aligned-by-row}(X,Y) \leftarrow \text{aligned-only-upper-row}(X,Y)$
 $\text{aligned-by-row}(X,Y) \leftarrow \text{aligned-only-middle-row}(X,Y)$
 $\text{aligned-by-row}(X,Y) \leftarrow \text{aligned-only-lower-row}(X,Y)$
 $\text{aligned-by-row}(X,Y) \leftarrow \text{aligned-both-rows}(X,Y)$

Moreover it is possible to define the following predicates:

$\text{aligned-left-col}(X,Y) \leftarrow \text{aligned-only-left-col}(X,Y)$
 $\text{aligned-left-col}(X,Y) \leftarrow \text{aligned-both-columns}(X,Y)$
 $\text{aligned-middle-col}(X,Y) \leftarrow$
 $\text{aligned-only-middle-col}(X,Y)$
 $\text{aligned-middle-col}(X,Y) \leftarrow \text{aligned-both-columns}(X,Y)$
 $\text{aligned-right-col}(X,Y) \leftarrow \text{aligned-only-right-col}(X,Y)$
 $\text{aligned-right-col}(X,Y) \leftarrow \text{aligned-both-columns}(X,Y)$
 $\text{aligned-middle-row}(X,Y) \leftarrow$

$\text{logic_type-sender}(x2) \leftarrow$

$\text{logic_type-receiver}(x3), \text{logic_type-unsigned}(x4),$
 $\text{logic_type-logo}(x5), \text{logic_type-date}(x6), \text{logic_type-ref}(x7),$
 $\text{logic_type-unsigned}(x8), \text{logic_type-unsigned}(x9),$
 $\text{logic_type-unsigned}(x10), \text{logic_type-unsigned}(x11),$
 $\text{part_of}(x1,x2), \text{part_of}(x1,x3), \text{part_of}(x1,x4), \text{part_of}(x1,x5),$
 $\text{part_of}(x1,x6), \text{part_of}(x1,x7), \text{part_of}(x1,x8), \text{part_of}(x1,x9),$
 $\text{part_of}(x1,x10), \text{part_of}(x1,x11),$
 $\text{width-medium}(x2), \text{width-medium-large}(x3),$
 $\text{width-smallest}(x4), \text{width-medium}(x5),$
 $\text{width-medium-small}(x6), \text{width-medium-large}(x7),$
 $\text{width-very-very-large}(x8), \text{width-very-very-large}(x9),$
 $\text{width-medium-large}(x10), \text{width-smallest}(x11),$
 $\text{height-medium-large}(x2), \text{height-small}(x3),$
 $\text{height-smallest}(x4), \text{height-very-small}(x5),$
 $\text{height-very-very-small}(x6), \text{height-very-very-small}(x7),$
 $\text{height-smallest}(x8), \text{height-large}(x9),$
 $\text{height-medium-small}(x10), \text{height-smallest}(x11),$
 $\text{type-text}(x2), \text{type-text}(x3), \text{type-text}(x4),$
 $\text{type-picture}(x5), \text{type-text}(x6), \text{type-text}(x7),$
 $\text{type-text}(x8), \text{type-text}(x9), \text{type-mixture}(x10),$
 $\text{type-text}(x11), \text{position-top-left}(x2), \text{position-top}(x3),$
 $\text{position-top-left}(x4), \text{position-top-left}(x5),$
 $\text{position-top-right}(x6), \text{position-top}(x7),$
 $\text{position-center}(x8), \text{position-center}(x9),$
 $\text{position-bottom-right}(x10), \text{position-bottom-left}(x11),$
 $\text{on_top}(x5,x8), \text{on_top}(x6,x8), \text{on_top}(x7,x8), \text{on_top}(x9,x10),$
 $\text{to_right}(x2,x4), \text{to_right}(x5,x7),$
 $\text{aligned-both-columns}(x2,x5), \text{aligned-only-lower-row}(x5,x7),$
 $\text{aligned-only-left-col}(x4,x7), \text{aligned-both-rows}(x7,x6),$
 $\text{aligned-only-right-col}(x8,x9), \text{aligned-only-upper-row}(x4,x3),$
 $\text{aligned-only-left-col}(x8,x11)$

Figure 4. Ground Horn clauses for the sender of the layout in Figure 3.

$\text{aligned-only-middle-row}(X,Y)$
 $\text{aligned-middle-row}(X,Y) \leftarrow \text{aligned-both-rows}(X,Y)$
 $\text{aligned-upper-row}(X,Y) \leftarrow \text{aligned-only-upper-row}(X,Y)$
 $\text{aligned-upper-row}(X,Y) \leftarrow \text{aligned-both-rows}(X,Y)$
 $\text{aligned-lower-row}(X,Y) \leftarrow \text{aligned-only-lower-row}(X,Y)$
 $\text{aligned-lower-row}(X,Y) \leftarrow \text{aligned-both-rows}(X,Y)$

It is worthwhile to notice that *aligned-by-column*(X,Y) means that X and Y are aligned by column and X is above Y. However, this does not imply that X is *on_top* Y, since the literal *on_top*(X,Y) states that X is above Y and their distance is less than 50 points on the vertical axis. Thus, it makes sense to define the predicate *above* as follows:

$\text{above}(X,Y) \leftarrow \text{on_top}(X,Y)$
 $\text{above}(X,Y) \leftarrow \text{aligned-by-column}(X,Y)$

Analogously, we defined the predicate *to_the_right_side* as follows:

$\text{to_the_right_side}(X,Y) \leftarrow \text{to_right}(X,Y)$

to_the_right_side(X,Y) \leftarrow aligned-by-row(X,Y)

Horn clauses allow recursion to be represented as well. Recursion can be quite useful for the application of document understanding. For instance, let us consider the portion of layout shown in Figure 5. For some reason, it happened that the logical component sender has been fragmented into several layout blocks, but the fragment 4 can be easily recognized since it is above a block of type picture. If recursive definitions are used, then we can easily label blocks 1, 2 and 4 by means of the following rule:

sender(X) \leftarrow above(X,Y), type-picture(Y)

sender(X) \leftarrow on_top(X,Y), sender(Y).

Any other rule for the recognition of blocks sender would be more complex, and, in our opinion, sometimes less accurate than that given above. However, recursion should be used with caution: it is necessary to check the existence of a termination condition in all cases in order to avoid infinite recursion. For instance, the recursive rule:

logic_type-ref(X) \leftarrow to_the_right_side(X,Y),
to_the_right_side(Z,Y),
logic_type-ref(Z),
aligned-upper-row(X,W)
logic_type-ref(X) \leftarrow width-small(X),
central-vert-band(X)

can cause infinite recursion since X and Z can be bound to the same layout block.

There are several types of recursion. In *direct* recursion, like the previous example, the same literal appears both in the head and in the body of a clause. In *indirect* (or *mutual*) recursion, a predicate p(x) in the body of a clause whose head

is q(x) appears in the head of another clause whose body contains q(x):

q(x) \leftarrow p(x), r(x) p(x) \leftarrow q(x), s(x).

In *tail* recursion, the computation of a function has already been completed when the axiomatic level is reached. An example of tail recursion is given by the following definition of the Prolog predicate *reverse*:

reverse(X,Y) \leftarrow reverse1(X,[],Y)

reverse1([],X,X)

reverse1([A|X],Y,Z) \leftarrow reverse1(X,[A|Y],Z)

which is more efficient than the classical direct-recursive definition:

reverse([],[])

reverse([A|X],Y) \leftarrow reverse(X,Z), append(Z,[A],Y).

In fact, tail-recursion generally uses memory more efficiently than direct recursion and therefore it would be better to learn, when possible, tail-recursive definitions. However, this means that new predicates, which were not present in the original definition, have to be generated (e.g., *reverse1*), thus the problem is further complicated but the solution lies in constructive induction.

For the application of document understanding, direct recursion is the most useful form of recursion we need, even if in learning dependent concepts indirect recursion is also desirable. We use an option of FOCL that implements a very simple technique to prevent infinitely recursive clauses. In particular, when a recursive literal p(Y) is added to the body of a clause whose head is p(X), the literal not(X=Y) is added as well. This technique is better than that implemented in mFOIL (Dzeroski and Bratko, 1992), but less sophisticated than that reported in (Quinlan, 1990) for FOIL.

3. An Algorithm That Learns Horn Clauses: FOCL

FOCL (Pazzani and Kibler, 1992) is an extension of FOIL (Quinlan, 1990) in several aspects. As FOIL, it implements a separate-and-conquer

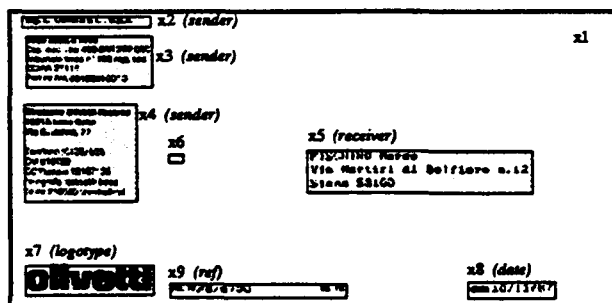


Figure 5. The problem of fragmentation in page layout.

strategy to learn a rule. Given a set of positive and negative instances of a concept $p(X_1, X_2, \dots, X_n)$ to learn, FOCL starts with the initial hypotheses:

$$p(X_1, X_2, \dots, X_n) \leftarrow$$

whose body is empty and repeatedly adds a new literal $q(Y_1, Y_2, \dots, Y_k)$ until the clause covers only positive instances. Given a clause:

$$p(X_1, X_2, \dots, X_n) \leftarrow \phi(Z_1, Z_2, \dots, Z_m)$$

where $\phi(Z_1, Z_2, \dots, Z_m)$ denotes a conjunction of literals, a *tuple* is a value assignment for the variables $X_1, X_2, \dots, X_n, Z_1, Z_2, \dots, Z_m$ such that the clause is satisfied. In particular the tuple is positive/negative if the value assignment for the variables in the head coincides with a positive/negative instance of the predicate $p(X_1, X_2, \dots, X_n)$. The addition of a literal $q(Y_1, Y_2, \dots, Y_k)$ to the body of an inconsistent clause may change the set of tuples covered by the clause and, consequently, the proportion of positive and negative tuples in such a set. Among all possible literals that can be added to a clause, that one maximizing an information theoretic heuristic, called *information gain*, is selected. In fact, a positive information gain means that the proportion of positive tuples with respect to the set of covered tuples is increased by adding a given literal to the clause. The information gain can be computed for predicates defined both extensionally and intensionally. When a partial, possibly incorrect, intensional definition of the concept to learn is provided, FOCL uses the information gain metric in order to operationalize the concept description as in explanation-based learning. However, in our application to document understanding such a potential is not exploited.

Another characteristic that distinguishes FOCL from FOIL is the availability of *relational clichés* that suggests potentially useful combinations of predicates to test while generating a clause of a predicate definition (Silverstein and Pazzani, 1991). In this way, clichés provide a form of look-ahead that tries to overcome the problem of horizon effect leading

a hill-climbing search strategy to find local rather than global maxima. For instance, the following clause:

$\text{logic_type_sender}(X) \leftarrow \text{above}(X, Y), \text{type_picture}(Y)$

that allows several logical objects of type sender to be recognized, cannot be generated without the introduction of clichés, since the predicate $\text{above}(X, Y)$ has a very small positive information gain and FOCL prefers other literals to that. In this case, it is the type of the block below a sender to discriminate a sender from any other kind of logical object, but unfortunately, throwing out the literal $\text{above}(X, Y)$, prevents the learning system from discovering such a discriminant information. By introducing a relational cliché in which FOCL is said to test the couples of literals:

$\text{above}(X, Y), q(Y_1, Y_2, \dots, Y_k)$

the above inconvenience is solved, but the search space to explore becomes wider.

4. Experimental Results

In section 2, a set of inference rules that form the background knowledge has been defined. However, the utility of these rules has to be evaluated empirically. For this reason, we considered a set of 30 single page documents, namely copies of letters sent by Olivetti. For each experiment, this set was randomly split into two subsets according to the following criterion:

- 20 documents for the training set
- 10 documents for the test set.

There are five concepts to learn, namely sender of the letter, receiver, logotype, reference number and date (other concepts, such as body of the letter or signature will be considered in future experiments). Obviously, not all blocks are instances of one of these concepts, that is there are some unlabelled blocks that we are not interested to classify. Moreover, there might be more than one block with the same label in a document, since some logical components might have been fragmented into several layout blocks

that the layout analysis was not able to group together.

In the first experimentation no background knowledge was used during the generalization process. Six different experiments were organized by randomly selecting the documents for the training and test sets. Results for each experiment are reported in Table II, where the entries n/m of each experiment report the number of commission (n) and omission (m) errors for each rule. The last column of the table reports the average of the error rates for each experiment computed as the sum of commission and omission errors divided by the number of logical components in the test set for a given class. The TOTAL average error is calculated as the average of the TOTAL errors for each experiment, divided by the number of logical components (including unlabelled blocks). The entry "tot. cost" for each experiment concerns the number of tested literals, namely those involving intensionally and extensionally defined predicates, as well as the pairs of literals tested with clichés. Since in this first experimentation neither background knowledge nor relational clichés were used, the reported numbers refer to the number of literals involving only extensionally defined predicates.

Table III summarizes results concerning the second experimentation in which the background knowledge is used. The structure of the table is the same as that of Table II, but in this case the total cost includes the number of literals involving intensionally defined predicates as well. It is

Table II
Results of the first experimentation:
basic case

rule/exp	1	2	3	4	5	6	av. error
logo	0/0	0/0	0/0	0/0	0/0	0/0	0.0 %
sender	0/0	0/1	0/1	0/0	0/9	0/1	14.1 %
ref	0/0	2/0	0/1	0/0	3/0	0/2	6.9 %
date	0/2	1/0	0/0	0/2	6/2	2/4	25.4 %
receiver	2/1	5/1	0/0	3/2	3/5	2/3	40.1 %
TOTAL	2/3	8/2	0/2	3/4	12/16	4/10	8.3 %
tot. cost	5544	2924	6787	7217	5560	5113	33145

possible to note that the introduction of the background knowledge did not significantly improve the predictive accuracy of the final rules since the difference is only .2% better than the basic case, while the number of the tested literals varies from 36% to 300% more than that for the basic case.

In the third experimentation two clichés were introduced, namely the *ON_TOP* cliché for the pairs of literals

$\text{on_top}(X,Y), q(Y_1, Y_2, \dots, Y_p)$

in which q is a predicate defined either extensionally or intensionally and at least one of the variables Y_i is in the set $\{X,Y\}$, and the *TO_RIGHT* cliché that tested all pairs of literals

$\text{to_right}(X,Y), q(Y_1, Y_2, \dots, Y_p)$.

By looking at the results reported in Table IV, it is possible to observe that the introduction of clichés did not improve significantly the accuracy with respect to the basic case (no background knowledge and no clichés). Strangely, it seems that by enlarging the search space helps to learn better rules for some class, such as *ref*, but it leads the system to consider wrong hypotheses for other concepts, such as *date*.

In another experimentation (see Table V) both the background knowledge and four clichés were introduced. In particular, in addition to *ON_TOP* and *TO_RIGHT*, other two clichés were introduced, namely *ABOVE* and *TO_THE_RIGHT_SIDE* that allow for testing pairs of literals in which the first literal contains the intensionally defined predicates *above* and

Table III
Results of the second experimentation:
basic case + BK

rule/exp	1	2	3	4	5	6	av. error
logo	0/0	0/0	0/0	0/0	0/0	0/0	0.0 %
sender	0/0	0/1	0/1	0/0	0/9	0/1	14.1 %
ref	0/0	1/2	0/1	0/0	5/0	0/0	10.0 %
date	1/2	0/0	0/0	0/2	5/3	2/4	25.1 %
receiver	3/1	2/1	0/0	2/1	0/5	2/3	36.1 %
TOTAL	4/3	3/4	0/2	2/3	10/17	4/8	8.1 %
tot. cost	9084	8750	10693	9547	6457	6998	51529

to_the_right_side respectively. In this experimentation the global results worsen (average error rate = 8.5) and for one training set FOCL was not able to generate a complete and consistent rule for the concept *date* after almost 4 hours of CPU time on a SUN station 4/25. Indeed, FOCL preferred to introduce predicates with new variables (more than 10), so enormously widening the search space at each step without really improving consistency. The existence of a simpler and consistent rule is guaranteed by the fact that in the previous experiments with only 2 clichés FOCL always converged towards a solution. A way to force FOCL to converge towards a simpler solution than that it is looking for is to define a limit on the maximum number of new variables that can be introduced in a rule. However, this is simply a trick to bypass the problem of divergence, but the true problem is in the information gain function that guides the hill-climbing search. Indeed, this heuristic seems

biased towards the introduction of new variables rather than towards the discriminatory power of a literal. For instance, in a run we observed that the system preferred the pair of literals:

to-the-right-side(X, Y), *aligned-upper-row*(Z, Y) covering 34/36 positive tuples and 38/79 negative tuples (gain 67.0), rather than the literals:

to-right(X, Y), *logic_type-logic*(Y)

that covered 21/25 positive tuples and 1/42 negative tuples (gain 62.7). A similar problem on the gain function used in ID3 for the selection of the next attribute to test has also been noticed by Fayyad in the induction of decision trees for multiple concept learning (Fayyad, 1991).

Since in section 2 the generation of recursive rules has been claimed useful for the problem of document understanding, we also tried to introduce recursion in the learning process. Table VI summarizes the experimental results, which would be probably better than those shown in Table V had not the concept *date* led to problems of non convergence of the learning process and infinite recursion. In fact, in one case FOCL did not generate any rule after 4 hours of CPU time, while in another it generated the following rule:

logic_type-date(X) \leftarrow *to_the_right_side*(Y, X),
 to_the_right_side(Z, Y),
 above(X, W), *height-smallest*(W),
 \neg *to_right*(Y, X).
logic_type-date(X) \leftarrow *to_the_right_side*(Y, X),
 \neg (X = Y), *logic_type-date*(Y).
logic_type-date(X) \leftarrow *to_the_right_side*(Y, X),
 to_the_right_side(Z, Y),

Table IV

**Results of the third experimentation:
 basic case + two clichés**

rule/exp	1	2	3	4	5	6	av. error
logo	0/0	0/0	0/0	0/0	0/0	0/0	0.0 %
sender	0/0	0/1	0/1	0/0	0/9	0/1	14.1 %
ref	0/0	0/1	0/1	1/0	0/0	0/0	2.8 %
date	1/3	3/0	0/0	0/4	6/2	2/4	30.0 %
receiver	2/0	0/1	0/0	3/1	3/5	8/3	36.9 %
TOTAL	3/3	0/3	0/2	4/5	9/16	10/8	8.0 %
tot. cost	10428	8636	8421	10695	5979	8212	52371

Table V

**Results of the fourth experiment:
 basic case + 4 clichés + BK**

rule/exp	1	2	3	4	5	6	av. error
logo	0/0	0/0	0/0	0/0	0/0	0/0	0.0 %
sender	0/0	0/1	1/1	3/0	0/9	0/3	22.2 %
ref	0/0	0/1	0/1	2/0	9/0	0/0	15.4 %
date	0/1	3/0	0/0	4/0	7/1	-	27.3 %
receiver	1/0	0/1	1/1	2/0	3/4	0/1	20.0 %
TOTAL	1/1	3/3	2/3	11/0	19/14	0/4	8.5 %
tot. cost	38957	38999	42346	31370	12469	16617	180758

Table VI

**Results of the fifth experimentation:
 basic case + 4 clichés + BK + recursion**

rule/exp	1	2	3	4	5	6	av. error
logo	0/0	0/0	0/0	0/0	0/0	0/0	0.0 %
sender	0/0	0/1	0/1	0/0	0/9	0/1	14.1 %
ref	0/0	0/1	0/1	2/0	9/0	0/0	15.4 %
date	0/1	-	2/2	6/0	7/1	-	42.3 %
receiver	2/0	0/1	2/1	2/0	3/4	1/0	22.0 %
TOTAL	2/1	0/3	4/5	10/0	19/14	1/1	10.3 %
tot. cost	34843	33403	29653	33078	12635	16793	160405

```

to_the_right_side(Z, W),
central-vert-band(W),
above(X, V),
aligned-by-column(V, U),
logic_type-date(X) ← to_the_right_side(Y, X),
aligned-lower-row(Z, Y),
¬to-right(Y, X),
width-medium-small(X),
logic_type-date(X) ← to_right(X, Y), ¬(X = Y),
logic_type-date(Y),
width-small(X).

```

which causes problems of infinite recursion for some block, say B1, which is not *to_the_right_side* of another block but it is itself *to_right* another block, say B2. Indeed, in this case the first four clauses fail to prove the goal *logic_type-date(B1)*, while the last clause is satisfied if it is proven the goal *logic_type-date(B2)*. However, if the first clause cannot explain this subgoal, the classifier finds the second clause in which the first literal is certainly true, and then tries to prove again the goal *logic_type-date(B1)*, thus falling in infinite recursion.

5. Contextual Learning

5.1 The problem

Inductive learning is undoubtedly the paradigm that has been most widely investigated in machine learning. In particular, several studies have been made on learning from examples and many learning systems have been developed. They differ in several aspects, such as the *representation language* used to represent examples or observations as well as the background knowledge and the hypotheses, the *search strategy* adopted to search in the space of hypotheses defined by the representation language, the amount and the type of *background knowledge* exploited during the learning process.

However, there is an aspect that joins most of the studies on learning from examples: *the basic assumption that concepts are mutually independent*. Even though in many applications

such an assumption is reasonable, this is not generally true. For instance, if our aim is that of recognizing flowers and trees in a picture, we can try to learn the concepts independently. This means that we provide a learning system with instances of trees and flowers and we try to find those properties that characterize them. However, in this case we are deliberately neglecting all other properties that relate the two concepts, such as their relative height (trees are taller than flowers), that can make easier the recognition task in most of the cases. Astute readers will surely note that relationships between the concepts of trees and flowers do not *characterize* the two concepts, but express a *constraint* between them. In fact, when the *independence assumption* is not made, the learning problem becomes that of learning properties that characterize each concept (or discriminate it from other concepts) as well as *dependencies* (or constraints) between concepts.

A natural consequence of concept dependency is that *instances of dependent concepts are dependent themselves*. This gives us an immediate way to recognize those learning systems in which the independence assumption is made: they allow for representing only independent instances. For example, all relational databases used by the machine learning community to test different learning systems represent instances of concepts as $(n+1)$ -tuples:

$$\langle a_1, a_2, \dots, a_n, c \rangle$$

where a_i are attributes of the concept and c is the membership class (i.e. the name of the concept). In this case concept dependencies can only be expressed by allowing the domain of some attribute a_i to contain values of the domain of the class attribute c , but this is never done. Sometimes, the assumption that classes are *mutually exclusive* is made explicit (Quinlan, 1986), while rarely the problem of instances that belong to different classes (e.g. patients that show symptoms common to two different diseases) is at least considered (Michalski, 1983).

Another clear example of the independence

assumption can be found in the parametric methods studied in statistical pattern recognition. Indeed, the formulation of these methods begins with the statement that each class is described by a *class distribution* function, $p(x_i | x_i \in C_j, \theta_j)$, which gives the probability of a datum x_i if it were known to belong to class C_j (θ_j is simply a class parameter vector that characterizes the class distribution in a parametric family of distributions) (Hand, 1981). Trivially, in the case of diseases that show common symptoms, class distribution functions are no longer adequate and we need to consider the joint probability function $p(x_i | x_i \in C_1, x_i \in C_2, \dots, x_i \in C_k, \theta)$.

It should be observed that the fact that instances are considered independent by all the methods of learning from examples does not mean that concepts are really independent. Indeed, independence is only an assumption, that can be adequate or not according to the problem at hand. This means that it is also possible to learn dependencies between concepts even though instances show no explicit form of such dependencies. A typical example of learning dependencies between concepts is met in the area of statistical causal inference (Esposito *et al.*, 1993a).

When concept dependencies are intrinsically acyclic. In this case we can use *dependence hierarchies*, that is directed acyclic graphs whose nodes are concepts to learn, to represent them (see Figure 6). The order in which concepts should be learned is completely defined by the dependence hierarchy, in particular the concepts in the lowest level of a dependency hierarchy have to be learned first, since their definition

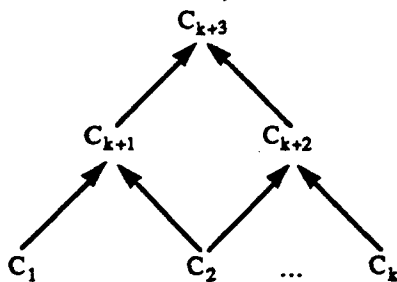


Figure 6. An example of dependence hierarchy between concepts

does not depend on other concepts. Some authors name such concepts *golden points* (Baroglio and Giordana, 1992) but we prefer the term *minimally dependent* concepts. In some studies on inductive learning, the necessity to learn dependent concepts has been implicitly recognized. For instance, many studies on inductive logic programming have reported Shapiro's problem of learning two Prolog predicates, *append* and *reverse* (Shapiro, 1981). Obviously, in all these studies the predicate *reverse* was learned only after that the predicate *append* had been learned. For instance, FOIL can learn the predicate *append* in 57.9 secs and then requires 29.0 secs for learning the predicate *reverse* on a SUN station 4/25. Not surprisingly, if the concepts are learned independently, the rule produced for the predicate *reverse* is no longer correct and the time required for learning is about 78.2 secs. This is another proof, if necessary, of the problems with accuracy of hypothesis and efficiency of the learning process that the independence assumption can cause.

5.2 Contextual problems in structural domains

The problem of contextual learning has its natural setting in structural domains, that is those domains in which concepts, observations and background knowledge can be effectively described by means of first-order logic rather than propositional calculus. The need to move to a more powerful representation language is that observations can be made up of several parts, thus the representation of one of such observations consists of a number of attributes concerning each subpart as well as of different relationships between subparts.

Relationships can be easily represented as first-order logic predicates, but can hardly be expressed by means of propositional calculus (Quinlan, 1990). There are several examples of structural domains presented in the machine learning literature. Winston's arch problem

(1970) and Michalski's problem of trains going east or west (1980) are just some of them. A common aspect to all these problems is that we want to predict a property, the class, that concerns the objects as a whole. However, there are a number of problems in which rules that allow some subparts of an object to be classified are sought. Many of such problems can be easily found in the literature on computer vision. Here are two of them:

- Scene labelling problem: Given a picture taken from a scene, say an office, it is required to identify some objects in the scene. A low-level computer vision system preprocess the picture, segments it and generates numerical or symbolic features for each segment as well as a description of how segments are related (for instance, a segment is included into another, or is adjacent to another). At this stage, it is necessary to associate each or some segments with the name of the object in the scene they represent. The problem is further complicated by the fact that the picture of an object can be fragmented into several segments.
- Edge orientation problem: Given a picture of one or more objects, in which the edges of the objects have already been detected, it is required to identify the orientation of each segment (or line) composing the edges. This information can help to reconstruct the 3-D scene and to understand how objects are related in terms of 3-D features (for instance, an object is in front of another one). Typical examples of line labels are: border, convex and concave edge.

Other problems of this kind, which are named *labelling problems*, can be found in the area of speech recognition (identification of words or phonemes in an acoustic signal), fault diagnosis as well as in the game theory.

Generally speaking, in a labelling problem we are given a complex object O which can be decomposed in a set $U = \{u_1, \dots, u_n\}$ of *units* each of which can be named with a *label* l_i taken from

a set $L = \{l_1, \dots, l_m\}$ of labels. The description of O is given by the description of each single unit in terms of a set of attributes $A = \{a_1, \dots, a_p\}$ as well as by the description of their relationships in terms of a set of relations $R = \{r_1, \dots, r_q\}$. The aim is that of associating the right label to some (or all) units of O . As said above, structured objects can be easily represented in first-order logic. For instance, by assuming that all relations in R are binary, we can represent O as a conjunction of (typically positive) literals involving binary predicates:

$$a_1(u_1, c_{11}) \wedge a_1(u_2, c_{12}) \wedge \dots \wedge a_1(u_n, c_{1n}) \wedge \dots \wedge a_p(u_n, c_{pn}) \\ \wedge r_1(u_1, u_2) \wedge \dots \wedge r_q(u_{q-1}, u_q)$$

where c_{ij} are constants representing values of the attributes while u_i are constants used to denote each object. Obviously, when labels are known, the above conjunction will be conjoint with as many literals $l(u_i, l_{ij})$ as the number of labelled units (here l is the *label* predicate and the constants l_{ij} are elements of L). Given a set of instances of a structured object O for which some (or all) units have been labelled, the learning problem is that of learning rules that allow for labelling their units. For instance, given a set of pictures of an office in which we have labelled the segments with the name of some objects, such as chair, desk and so on, we want to discover some rules for labelling segments so that we will be able to recognize objects in a new picture.

Typically, in labelling problems it is not convenient to learn concepts independently of each other since concepts are themselves strongly related or mutually constrained. The independence assumption on which most learning algorithms rely is not generally adequate, and it may lead to learn as many rules as the number of instances of a given concept.

Luckily, in some labelling problems, it may happen that concept dependencies are intrinsically acyclic and it is possible to define in a quite natural way a dependence hierarchy. In this case it is possible to start the learning process with those concepts in the hierarchy that appear at the very bottom (minimally dependent

concepts). The language of observations will contain all *operational* (or extensionally defined) predicates, a_i and r_j (ground literals), while the language of hypotheses can be allowed to contain *non-operational* (or intensionally defined) predicates as well. It is worthwhile to observe that minimally dependent concepts are learned independently of each other, thus traditional learning systems that induce structural descriptions can be effectively employed in this step. Generalizations of the learned concepts, say C_1, C_2, \dots, C_k , will be a set of Horn clauses of the type:

$$l(X, l_1) \leftarrow \phi(X, Z_1, Z_2, \dots, Z_m)$$

where $\phi(X, Z_1, Z_2, \dots, Z_m)$ denotes a conjunction of literals concerning operational and non-operational predicates in which X, Z_1, Z_2, \dots, Z_m are the only variables that occur.

When the minimally dependent concepts have been learned, it is possible to learn those concepts that depend directly on C_1, C_2, \dots, C_k , but now the language of observations will contain the predicate l whose extensional definition is given by all possible instantiations $l(u_i, l_1), i \in \{1, 2, \dots, k\}$, used as positive examples in the previous step. If the dependence hierarchy is well defined, the rules produced for concepts learned in this step will contain at least one occurrence of the predicate l . More precisely, if the concept C_{k+1} directly depends on the concept C_1 and C_2 , we expect that the rule for C_{k+1} will contain at least one literal of the set $\{l(Y, l_1), \neg l(Y, l_1)\}$ as well as one literal of the set $\{l(Y, l_2), \neg l(Y, l_2)\}$. Such an expectation is explained by the fact that concepts C_1 and C_2 are useful to explain or predict C_{k+1} . When previous conditions do not hold, it means that either the dependence hierarchy is partly over-specific (some concept dependencies do not occur in reality) or evidence in the observations is not enough to detect such dependencies (this is particularly true when dependencies are probabilistic rather than deterministic) or the search strategy of the learning system is simply limited in some way. The multistep learning process continues until

all concepts in the dependence hierarchy have been learned.

5.3 Application to document understanding

In previous experiments concepts have been assumed to be independent. In this section we will show some experimental results of the attempt to learn contextual rules. The dependence hierarchy between concepts is reported in Figure 7. The reason for this definition can be partly explained in terms of spatial reasoning. In fact, when the logotype has been recognized, the recognition of the contiguous logical components should be easier. In this case the contiguous logical components are *sender* (which is above the logotype) and *ref* which is to the right of the *logotype*. When the *ref* has been recognized then the *date*, which is to its right, should be more easily recognized. Finally, when *sender*, *ref* and *date* are recognized the identification of the *receiver* should be easier. The reason for which the *logotype* has been chosen as minimally dependent concept is that the logotype is generally the only block of type picture and, moreover, it is never (or rarely) fragmented or grouped with other blocks, thus it should be the simplest concept to learn. After having defined a dependence hierarchy, FOCL was used to generate the rule for the concept *logotype* by using only the original set of predicates (extensionally and intensionally defined). Then, we ran FOCL to generate the rules for the concepts *sender* and *ref* by using the original set of predicates together with the predicate *learned_description_of_logic_type-logo* previously learned. Then, FOCL is asked to generate the rule for the concept *date* by using the original predicates plus *learned_description_of_logic_type-logo*,

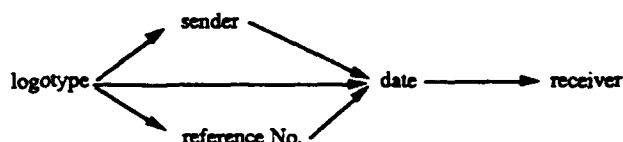


Figure 7. The hierarchy of concept dependencies

learned_description_of_logic_type-sender and *learned_description_of_logic_type-ref*. Finally, FOCL learned the concept *receiver* from both the original predicates and all the learned predicates.

Experimental results are shown in Table VII. First of all it should be noticed that the average error rate is 6.4%, that is the lowest rate we got in all the experiments (about 2.0% less than the basic case). Such an increase of predictive accuracy is a counter-intuitive result, since when a prediction of a rule depends on the prediction made by another rule it may happen that an error in the first rule to fire is propagated to the second (dependent) rule. Thus, the average error rate for contextual rules should be higher than the error rate for independent rules. However, this problem does not occur when concepts are really dependent and the dependence hierarchy is well defined, since this error propagation effect should not occur frequently. Actually, this is what we observed in our experiments.

Secondly, we did not get problems of non-convergence or infinite recursion, since contextual rules are easier to learn. Indeed, the total number of tested literals is 124,929 which is less than the number of tested literals in the other experiments in which 4 clichés were used to generate rules. Such a decrease of learning time is counter-intuitive as well since for some concepts the search space is wider and not smaller due to the introduction of new predicates. The rationale for these results is that, when concepts

are really dependent, information on the context should help to learn more quickly.

Thirdly, sometimes, even if concepts are really dependent, FOCL is not able to capture such dependencies due to its own search strategy. Actually, this is a problem of any traditional learning system rather than a fault of FOCL. For instance, this problem can be observed in FOIL as well. Indeed, in order to express a dependence between two concepts, relations between variables representing different logical components have to be introduced in a clause (in the case of document understanding, those relations are geometrical relations such as *above*, *on-top*, etc.). Unfortunately, relations have quite often a small or even negative information gain if taken alone, so a greedy strategy like hill-climbing will never consider them, at least in the early steps of the generation of a hypothesis. As a consequence, dependence between concepts will not be considered in the first steps of the learning process and the final result will be strongly influenced by the initial choices. As already pointed out in section 3, such a problem can be partially solved by means of either *determinate literals* (Quinlan, 1991), as in FOIL or relational clichés as in FOCL. This is the main reason for which we still preferred to use clichés in this last experimentation even if previous experimental results were unsatisfactory when clichés were used.

6. Conclusions

In this paper, the problem of document understanding has been approached by means of inductive learning, in particular learning from examples. This approach represents a novelty in the field of document understanding and presents the advantages of generality and easy customizing of the document management system.

It is worthwhile to notice that in our approach the content of text blocks is not exploited to recognize each single logical component. On the contrary, the recognition of logical components

Table VII
Results of the sixth experimentation:
contextual learning

rule/exp	1	2	3	4	5	6	av. error
logo	0/0	0/0	0/0	0/0	0/0	0/0	0.0 %
sender	0/0	0/1	0/1	0/0	0/9	0/1	14.1 %
ref	0/0	0/1	0/1	2/0	9/0	0/0	15.4 %
date	0/0	3/0	0/0	0/0	7/0	0/0	13.6 %
receiver	2/0	0/1	2/1	1/0	3/4	0/3	22.8 %
TOTAL	2/0	3/3	2/3	3/0	19/13	0/4	6.4 %
tot. cost	28055	19371	19869	29415	11733	16486	124929

based on the layout structure of a document allows an OCR system to read only some specific components of interest for a particular application.

Given a set of documents, training examples are descriptions of some logical components of the documents. Therefore, the problem is that of learning recognition rules for subparts of the logical structure of a document. This problem is a particular kind of labelling problem in which subparts of a complex object have to be labelled. The second novelty of this paper is that of criticizing the independence assumption made by almost all learning systems and proposing a new learning strategy, named contextual learning, in which a dependence hierarchy of concepts is exploited to define both the order in which concepts should be learned and the proper observation language.

The paper empirically compares the traditional learning strategy in which the independence assumption is made with such a contextual learning strategy. For this reason, FOCL, a system that learns Horn clauses, was employed to generate the recognition rules for five logical components of a set of single page documents, namely copies of letters sent by an Italian company. Results concerning the first strategy show that the introduction of background knowledge, relational clichés and recursion does not significantly improve the predictive accuracy of the whole set of rules, even though some single concepts are better recognized by introducing one of such variants of the basic case. The definition of a hierarchy of concept dependencies, based on the spatial contiguity of logical components, allows FOCL to easily learn contextual rules whose average error is the lowest.

These results have spurred us to investigate the problem of learning dependencies between concepts, namely building a dependence hierarchy between concepts, which is presently provided by the teacher. For this purpose, we adopted the same multistrategy learning approach followed for the problem of document

classification to the problem of document understanding. In fact, for each block it is possible to define a set of numerical features describing geometrical characteristics of a block as well as percentage of black pixels, number of transitions black-white, and so on. Such features can be appropriately managed by a parametric classifier, namely Fisher's linear discriminant functions, whose results after the training phase can be exploited in several ways. One way consists in mapping the classification results of the discriminant functions into a new predicate which is in turn used by the system that learns Horn clauses. The other way is that of exploiting information on the discriminatory power of the parametric classifier in order to define the order in which concepts should be learned, starting with those concepts that can be easily recognized. Some preliminary results are presented in (Esposito *et al.*, 1993b).

Acknowledgments

The authors wish to thank Enrico Annese of Olivetti Systems & Networks for having made available the set of documents used in our experiments.

References

- Baroglio, C. and Giordana, A., "Learning Contextual Relations," *Proceedings of the Third Italian Workshop on Machine Learning*, Rome, Italy, May 7-8, 1992.
- Clark, K. L., Negation as Failure, in *Logic and Databases*, Gallaire, H., and Minker, J. (Eds.), New York: Plenum Press, 1978.
- Dzeroski, S., and Bratko I., "Handling Noise in Inductive Logic Programming," *Proceedings of the International Workshop on Inductive Logic Programming (ILP 92)*, Tokyo, Japan, June 1992.
- Esposito, F., Malerba, D., Semeraro, G., Annese, E., and Scafuro G., "Empirical learning methods

for digitized document recognition: an integrated approach to inductive generalization," *Proceedings of the 6th Conference on Artificial Intelligence Applications*, Santa Barbara, California, March 1990.

Esposito, F., Malerba, D., Semeraro, G., "A Comparison of Statistical Methods for Inferring Causation," *Proceedings of the 4th International Workshop on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, 4-6 January 1993a.

Esposito, F., Malerba, D. and Semeraro, G., "Contextual Supervised Learning for Document Understanding," submitted to *7th International Conference on Image Analysis and Processing*, Monopoli, Italy, September 1993b.

Fayyad, U., M., *On the induction of decision trees for multiple concept learning*, Doctoral dissertation, EECS Department, The University of Michigan, 1991.

Hand, D., J., *Discrimination and classification*, Wiley, London, 1981.

Helft, N., Inductive generalization: a logical framework, in *Progress in Machine Learning - Proceedings of EWSL 87: 2nd European Working Session on Learning*, Bratko, I., and Lavrac, N. (Eds), Bled, Yugoslavia, May 1987.

Horak, W., "Office Document Architecture and Office Document Interchange Formats: current status of international standardization," *IEEE Computer*, October 1985.

Michalski, R., S., "Pattern recognition as rule-guided inductive inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2, 4, July 1980.

Michalski, R., S., A theory and methodology of inductive learning, in *Machine Learning: An Artificial Intelligence Approach* Vol. 1, Michalski, R. S., Carbonell, J., and Mitchell, T. M. (Eds.), Tioga Publishing Company, Palo Alto, California, 1983.

Muggleton, S., Inductive Logic Programming, in *Inductive Logic Programming*, Muggleton, S. (Ed.), Academic Press, London, 1992.

Pazzani, M., and Kibler, D., "The utility of knowledge in inductive learning," *Machine Learning*, Vol. 9, No 1, June 1992.

Quinlan, J., R., "Induction of decision trees," *Machine Learning*, Vol. 1, No 1, 1986.

Quinlan, J., R., "Learning logical definitions from relations," *Machine Learning*, Vol. 5, No 3, 1990.

Quinlan, J., R., "Determinate literals in inductive logic programming," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Sydney, Australia, August 1991.

Silverstein, G., and Pazzani, M., "Relational clichés: constraining constructive induction during relational learning," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, Illinois, June 1991.

Shapiro, E., "Inductive inference of theories from facts," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, British Columbia, August 1991.

Tang, Y., Y., Suen, C., Y., Yan, C., D., and Cheriet, M., "Document analysis and understanding: a brief survey," *Proceedings of the First International Conference on Document Analysis and Recognition*, Saint-Malo, France, 1991.

Tsujimoto, S., and Asada, H., "Understanding multi-articled documents," *Proceedings of the 10th International Conference on Pattern Recognition*, Atlantic City, New Jersey, 1990.

Winston, P., H., Learning structural descriptions from examples, in *The psychology of computer vision*, Winston P. H. (Ed), McGraw Hill, New York, 1975.

Towards GEST-3D: Learning Relations in 3-D Shapes

Jakub Segen

AT&T Bell Laboratories, rm. 4E-632

Holmdel, N.J. 07733-3030

segen@vax135.att.com

Abstract

This paper describes work in progress towards a computer vision system, that learns to recognise three-dimensional nonrigid objects, and to locate and orient them in space. Similarly to its two-dimensional predecessor GEST, the system uses graphs to describe shape structure, and relies on a multistrategy approach to learn models of classes of 3-D shapes. A key new component of this system is a constructive induction method that discovers geometric relations among parts in a three-dimensional shape.

Key words: Computer vision, graph, clustering, 3-D shape recognition.

1 Introduction

Structural models composed of parts and relations have been used for more than 20 years as structural models of shape (Barrow *et al.*, 1972; Shapiro, 1980; Nackman, 1984). Such models have proved especially suitable in computer vision, where recognition and interpretation are the main goals of modeling, and it is likely that they will find use in other shape related fields, such as image compression or computer graphics.

Construction of structural models requires learning. A computer vision system that ap-

plies multistrategy learning to form structural models has been described in (Segen, 1993). This system, named GEST, describes structure using graphs. It learns models of nonrigid 2-D shapes from numerical examples, without user's help. Using these models it recognises 2-D objects in video input and computes their pose in real time. GEST works well enough that it is being used as an input device that allows one to control graphics applications with hand gestures.

The success of GEST in two dimensions gives incentive to research towards a multistrategy learning system that operates in a three-dimensional world: a GEST-3D. Its current status is described in this paper. A key new component of GEST-3D is a representation for 3-D geometric relations, that permits multiple types of parts and relations, and a constructive induction method that learns 3-D relations from data.

2 GEST-3D

GEST-3D will learn to recognise three-dimensional nonrigid objects from incomplete information, and to estimate object's pose, that is the location and orientation in 3-D space. This system will use a multistrategy learning approach, analogous to that in GEST, to construct models of 3-

D shape classes and to infer pose estimators. The learning module of GEST-3D, shown schematically in Figure 1, consists of three parts: Constructive Induction, Graph Learning, and Pose Learning. The first part uses a new constructive induction method to discover symbolic relations in a set of numerically represented instances of three-dimensional shapes. These relations are used as the basic primitives in the graph learning part, which builds a structural model for each class of 3-D shapes. The third part learns parametric functions for computing pose, using statistical methods of robust estimation. The results of all three learning parts are collected in a model library, which by an object recogniser to interpret three-dimensional shapes.

Since graph representation is independent of space dimension, the graph learning methods (Segen, 1990) used in GEST are taken with almost no changes. The two remaining parts are based on new research. The constructive induction approach used for learning 3-D relations is presented in the following sections of this paper. Methods used for learning three dimensional pose will be described separately.

3 Constructive Induction of Relations

A geometric relation represents a range of values of displacement and rotation between parts, or a relative pose. In a rigid object these values are nearly constant, but in a nonrigid object they can vary between different object instances, and range of variations can be different for different pairs of parts. The goal of the constructive induction process is to discover geometric relations in numerical data, and represent them as symbols.

This constructive induction method examines a set of training shapes for natural groupings of part types and their relative poses in pairs of nearby parts. Each identified group represents a geometric relation, involving two parts and a distribution of parameters of their relative pose. A pair of primitive parts joined by one of the discovered relations is then treated as a higher order part, and the grouping process repeats, giving rise to a hierarchy of parts and relations, similar to the hierarchy proposed in (Marr and Nishihara, 1978).

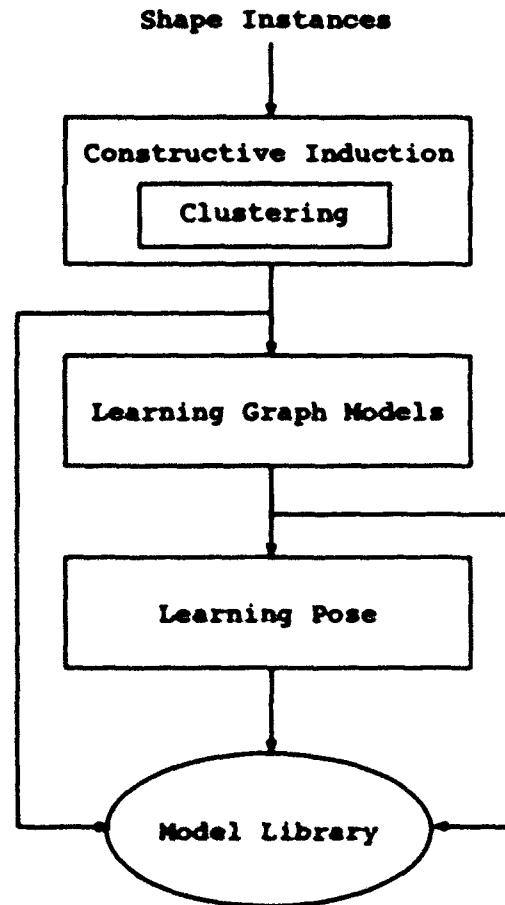


Figure 1: Learning in GEST-3D

4 Primitive Parts

An instance of shape is initially represented as a collection of primitive parts. These parts do not have to be mutually exclusive, that is some parts may overlap. The collection of parts does not have to cover the entire shape. The literature on shape provides an abundance of algorithms and techniques for extracting parts from a two- or three-dimensional shape. Some methods partition the primary representation, e.g. a curve, surface, or volume into homogeneous regions, i.e. regions whose local properties are approximately constant. Other methods identify singularities, that is points or boundaries of the primary representation which are unique within their neighborhood, such as edges, corners, local extrema of curvature, or critical points of a surface. Most of the published methods can be adapted to generate parts that satisfy the requirements of the approach described in this paper. This approach also allows one to mix together parts generated by different methods.

Definition: A primitive part p is a triple $[type, inv, var]$, where $type$ is a symbol from a finite alphabet, inv and var are real valued vectors, such that if T is a rigid transformation, then $T(p) = [type, inv, T(var)]$

The $type$ symbol specifies the format and interpretation for inv and var vectors, and it is used to distinguish parts generated by different extraction methods. The inv vector contains parameters that are invariant under rigid transformations of the coordinate system. These parameters are specific to the type of a part, which also remains constant under rigid transformations. Some parts, such as a single point, have no invariant parameters, that is the inv vector have dimension zero. The var vector consists of

parameters that change with rigid transformations. This vector carries information related to the part's pose, that is part's position and orientation in space. In 2-D space pose consists of three values: two coordinates of position and the orientation angle. In 3-D space pose has six dimensions: three position coordinates and three angles. If the pose of a part can be determined based on the form of the part, then the var vector is the pose. However, for parts with symmetries pose cannot be determined completely, but it can be restricted to a number of degrees of freedom (for continuous symmetries) or to a number of values (for discrete symmetries). The approach described here considers only parts with continuous symmetries such as line, or sphere. For such parts the var vector contains the maximal number of parameters that constrain the pose.

Definition: A representation by parts of shape S is a set of primitive parts $P(S) = \{p_1, p_2, \dots\}$.

The function P in this definition stands for a method used to segment or to extract parts from S . The representation $P(S)$ should satisfy the following properties:

1. **Invariance:** Without presence of noise a rigid transformation should not change the representation. This means that for any rigid transformation T , $P(S) = \{p_1, p_2, \dots\}$ implies $P(T(S)) = \{T(p_1), T(p_2), \dots\}$.
2. **Locality:** A part description should not be affected by shape changes outside of the part's immediate neighborhood.
3. **Stability:** The representation should not be significantly affected by random noise in the image. This means that

if S_1 and S_2 are two noisy images of the same shape, then there exists a one-to-one mapping between large subsets of $P(S_1)$ and $P(S_2)$, where the corresponding parts have identical type symbols, and similar *inv* and *var* parameters.

5 Constructing New Parts

All pairwise relations might contain useful information, but examining all such relations in a large set of parts can be too costly. Therefore, direct pairwise relations are restricted only to pairs of nearby parts. Pairs of nearby parts are also used to construct new parts, called *composite* parts, that are treated just like the primitive parts. One can find binary relations between composite parts, and combine a pair of composite parts into a new composite part.

One can think of a composite part as a root of a binary tree. All non-leaf nodes of this tree are composite parts, and the leaves are primitive parts. This binary tree defines the composite part at its root, and determines the order of operations needed to construct it. Depth of this tree determines the level of the part at the root. The level of a primitive part is 0, a level-1 part is constructed from a pair of primitive parts, two level-1 parts produce a level-2 part, and so on. The current method combines only parts with equal levels, so a composite part at level n has 2^n primitive parts at its leaves, i.e. it is a balanced tree. This restriction is used only for computational convenience, and it may be relaxed in the future.

A composite part is represented the same way as a primitive part as $[type, inv, var]$. Composite parts are constructed bottom-up, one level at a time up

to a preset maximal level.

The construction process repeats the following sequence of steps for each level:

1. Cluster the *inv* vectors of all parts at level k , separately for each part type, and assign a unique label to each cluster.
2. Assign to each part the label of its nearest cluster, or NIL if the nearest cluster is too far.
3. Terminate if k is equal to a preset maximal level.
4. Find pairs of neighboring parts with non NIL labels, and construct $k + 1$ level parts by applying a *composition* operation.

The result of the above construction process is a set of clusters *inv* vectors, grouped into part types. These clusters are saved in a library, which is later used by a recognition program to assign symbolic labels to primitive parts, and to relations among parts. The key element of the construction process is the composition operation (step 4) which forms new composite parts from pairs of existing parts. This operation is described in detail in the following section.

6 Composition of Parts

The composition operation is applied to an ordered pair of parts. The result of this operation is a composite part represented as $[type, inv, var]$.

The type of the result of composition is a string obtained by concatenating the types of components, treated as strings. The *inv* and *var* vectors of the result are computed from the *var* vectors of the components. This

operation depends on *symmetry types* of the component parts. A symmetry type specifies a group of rigid transformations which do not change the part's appearance. For example, an infinitely long cylinder has two continuous symmetries: rotation around, and translation along the axis. The approach proposed here is restricted to parts with continuous symmetries. Discrete symmetries, such as the symmetries of a cube may be treated in future extensions.

The symmetry type of a part is a function of the part type. The symmetry type of a primitive part has to be given defined for each primitive part type. Symmetry types of composite parts are determined by composition rules in Tables 2 and 4.

6.1 Parts in 3-D Space

Table 1 lists six symmetry types used for three-dimensional parts. The first two columns show a geometric form with a given type of symmetry, and part's symbol. Third column shows dimension of *var*. The symmetry transformations are shown in fourth column using a symbolic notation, where nR which means rotation about n axes, and nT means an n -dimensional translation. Table 2 shows the symmetry type and the number of invariant parameters of the result of composition, for all combinations of component symmetry types. In most cases it is possible to define the composition operation in several equivalent ways, using different expressions for the computing *inv* and *var* terms of the composition result from the components parts. The equivalence of such alternative formulations means that the composition result for one of the formulations contains enough information to compute the composition result of any other formulation, without using data from the com-

Table 1: Part types in 3-D

Part	Symb.	var	Symm.
Point	P	3	3R
Line	L	4	1R+1T
Plane	S	3	1R+1T
Point on line	PL	5	1R
Line on plane	LS	5	1T
Frame	F	6	None

ponent parts: Examples of definition of the composition operation are given below for four cases from Table 2. The following notation is used in these examples: $P(x)$ is the point specified by x , for symmetry types P and PL . $L(x)$ is the line and $r(x)$ the unit vector for symmetry types L , PL , and LS .

Case 1 Operands x, y , both of type P .

Result z of type PL :

var: $P(z)$ is the midpoint between $P(x)$ and $P(y)$, that is $\frac{1}{2}(P(x) + P(y))$. $L(z)$ is the line defined by $P(x)$ and $P(y)$.

inv: Distance between $P(x)$ and $P(y)$.

Case 2 Operands: x of type P , y of type PL .

Result z of type F :

var: The frame origin is the midpoint between $P(x)$ and $P(y)$. The orientation of the first axis is given by the unit vector

$$\vec{u} = \frac{P(x) - P(y)}{|P(x) - P(y)|}$$

The remaining axes are obtained by a Gram-Schmidt orthonormalization: Let $\vec{V} = r(y) - (r(y) \cdot \vec{u})\vec{u}$ then a unit vector $\vec{v} = \vec{V}/|\vec{V}|$ defines the second axis; the third axis is $\vec{w} = \vec{u} \times \vec{v}$.

inv: Distance between $P(x)$ and $L(y)$, and signed distance from $P(y)$ to the projection of $P(x)$ on $L(y)$.

Table 2: Part composition in 3-D

Comp.	Res.	inv
P,P	PL	1
P,L	F	1
P,S	PL	1
P,PL	F	2
P,LS	F	2
P,F	F	3
L,L	F	2
L,S	F	1
L,PL	F	3
L,LS	F	3
L,F	F	4
S,S	LS	1
S,PL	F	2
S,LS	F	2
S,F	F	3
PL,PL	F	4
PL,LS	F	4
PL,F	F	5
LS,LS	F	4
LS,F	F	5
F,F	F	6

Case 3 Operands: x, y , both of type PL .

Result z of type F :

var. Frame axes are computed as in Case 2, using $r(\tilde{x}) + r(\tilde{y})$ instead of $r(\tilde{y})$ to find the second axis.

inv. Distance between $P(x)$ and $P(y)$, angle between $r(\tilde{x})$ and \tilde{u} , angle between $r(\tilde{y})$ and \tilde{u} (\tilde{u} defined as in Case 2), and angle between the two planes defined by a line through $P(x)$ and $P(y)$ and vectors $r(\tilde{x})$, and $r(\tilde{y})$, or angle between normals to these planes.

Case 4 Operands: x, y , both of type F :

Result z of type F :

var. Frame axes are computed as in Case 1, using the sum of the unit vectors from all

Table 3: Part types in 2-D

Part	Symb.	var	Symm.
Point	P	2	1R
Line	L	2	1T
Point on line	PL	3	None

Table 4: Part composition in 2-D

Comp.	Res.	inv
P,P	PL	1
P,L	PL	1
P,PL	PL	2
L,L	PL	1
L,PL	PL	2
PL,PL	PL	3

six axes of z and y instead of $r(\tilde{y})$ to find the second axis. If this sum is 0 then any subset of the axes can be used.

inv. Six parameters of rigid transformation from x to y .

6.2 Parts in 2-D Space.

Symmetry types in 2-D form a subset of symmetry types in 3-D. This subset consists of three symmetry types listed in Table 3. The composition operation in 2-D is defined in Table 4. Each case of 2-D composition can be derived from a corresponding 3-D case, as shown in the example below.

Case 5 Operands: x, y , both of type PL in 2-D.

Result z of type PL :

var. $P(x)$ and $L(x)$ correspond to the origin and the first axis in Case 3 above.

inv. Distance between $P(x)$ and $P(y)$, and the two angles between vectors $r(x)$ and $r(y)$, and the line through $P(x)$ and $P(y)$.

7 Ordering Pairs of Parts

The procedure for matching the structural descriptions (Segen, 1990) requires the arguments of binary relations to be ordered. To order a pair of parts (x, y) the following three-step procedure is used. If x and y have different types then they are ordered according to a lexicographic order of their types. If the types are identical, but the parts have different labels, then they are ordered by their labels. If the labels are the same then an ordering function $f(x, y)$ is used. An ordering function must have the following properties:

1. There is a partial order relation $>$ defined on the range of f .
2. Generally, $f(x, y) \neq f(y, x)$.
3. For any rigid transformation T , $f(x, y) > f(y, x)$ implies $f(Tx, Ty) > f(Ty, Tx)$. Of course this is satisfied if $f(x, y) = f(Tx, Ty)$. This property ensures that the order specified by f is invariant under rigid transformations.

With the aid of f one orders parts x and y as (x, y) if $f(x, y) > f(y, x)$, and vice versa. If neither $f(x, y) > f(y, x)$ nor $f(y, x) > f(x, y)$, then parts cannot be ordered.

An example of such a function for 2-D parts x and y and symmetry types L , or PL , is the signed angle between part orientations $r(x)$ and $r(y)$.

An example for 3-D parts with symmetry type PL is the first angle invariant in Case 3 in Section 6.1.

Using an ordering function presents a minor problem. A natural cluster that intersects the hypersurface $f(x, y) = f(y, x)$ will be

split into two clusters. This is an undesirable feature, since such a split is purely artificial. Split clusters can be merged using the following consolidation procedure.

For each cluster C_i form an inverted cluster $-C_i$, then find a cluster C' in a set $C_i + 1, C_i + 2, \dots$, which is nearest to $-C_i$. An inverted cluster $-C$ is a cluster formed by reversing the order of composition of elements of C . In most cases such an operation is a function of cluster parameters, so it does not require reprocessing the cluster elements. If $-C_i$ and C' are sufficiently close then merge $-C_i$ into C' , provide a pointer from C_i to C' , and delete all clusters that point to C_i . In addition, delete any cluster that is close to its own inverse.

If the above procedure is used then the labeling step is modified as follows: If a composite part is assigned to a cluster a that points to a cluster b , then the part is inverted and receives the label of the cluster b .

8 Extracting Relations

The label of a primitive part symbolically describes an invariant property (unary relation), such as size or curvature. The label of a composite part P describes a binary relation between its two component parts or children. It also describes a 4th-order relation among the part's grandchildren (if any), 8th-order relation among the great grandchildren, and so on, until it finally describes a 2^n -ary relation over a set of primitive parts.

After constructing the composite parts up to a preset level, one retains only their labels, and the parent-child links. The resulting structure is a graph with labeled vertices, that are grouped into layers according to their depth. The leaves of the graph represent the primitive parts; other vertices rep-

resent composite parts. This graph contains all the information about the shape, that is used for recognition and interpretation.

9 Final Remarks

The relation constructor has been programmed only for the 2-D case, (Segen, 1998; 1989) and used as a module in the GEST system. This implementation uses one type of a primitive part: a local extremum of curvature of the boundary of a planar shape. This part has one invariant parameter: the curvature. The var vector contains the position of the extremal point, and the direction of the curve normal at this point, that is its symmetry type is *PL*. The symmetry type of all the composite parts derived from these primitives is also *PL*.

A first 3-D implementation will use corner-like parts of symmetry type *PL*, that are extracted by a structural stereo vision system.

References

Barrow, H. G., Ambler, A. P., Burstall, R. M., Some techniques for recognising structure in pictures. In *Frontiers of Pattern Recognition*, S. Watanabe (ed.), Academic Press, 1972.

Marr, D. Nishihara, H. K., Representation and recognition of spatial organisation of three-dimensional structures. *Proceedings of Royal Society*, B200, pp. 269-294, 1978.

Nackman, L. R., Two-dimensional critical point configuration graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 1984.

Segen, J., Learning graph models of shape. *Proceedings of the 5-th International Conference on Machine Learning*, pp. 29-35. Ann Arbor, MI, June, 1988.

Segen, J., Model Learning and Recognition

of Nonrigid Shapes. *Proceedings of Conference on Computer Vision and Pattern Recognition*, pp. 597-602, San Diego, CA, 1989.

Segen, J., Graph clustering and model learning by data compression. *Proceedings of the 7-th International Conference on Machine Learning*, pp. 93-101, Austin, Texas, 1990.

Segen, J., GEST: A learning computer vision system that recognises hand gestures. *Machine Learning IV*, R. S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, 1993.

Shapiro, L. G., A structural model of shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2, pp. 111-126, 1980.

Thinking and Seeing in Game Playing: Integrating Pattern Recognition and Symbolic Learning

Susan L. Epstein
Pascal Abadie
Joanna Lesniak
Department of Computer Science
Hunter College and The Graduate School
The City University of New York
sehhc@cunyvm.cuny.edu

Jack Gelfand
Department of Psychology
Frank Midgley
Department of Computer Science
Princeton University
Princeton, NJ
jjg@phoenix.princeton.edu

Abstract

Although people rely heavily on visual cues during problem solving, it is non-trivial to integrate them into machine learning. This paper reports on three general methods that smoothly and naturally incorporate visual cues into a hierarchical decision algorithm for game playing: two that interpret predrawn straight lines on the board, and a third that uses an associative, hierarchical pattern database for pattern recognition. They have been integrated into Hoyle, a game learning program that makes decisions with a hierarchy of modules representing individual rational and heuristic agents.

Key words: machine learning, game playing, hierarchical decision algorithms, visual cues, pattern recognition

1. Introduction

Since the early work of Chase and Simon, researchers have noted that expert chess players retain thousands of patterns (Holding, 1985).

There has been substantial additional work on having a program learn specific patterns for chess (Berliner, 1992; Campbell, 1988; Flann, 1992; Levinson and Snyder, 1991). There is conflicting evidence as to whether or not expert game players learn to play solely by associating appropriate moves with key patterns detected on the board, but it is believed that pattern recognition is an important part of a number of different strategies exercised in expert play (Holding, 1985). In AI, visual cues have previously demonstrated their power as explicit search control directives and as hand-selected terms in an evaluation function (Gelernter, 1963; Samuel, 1963). Learned visual cues have also been derived from goal states with a predicate calculus representation (Fawcett and Utgoff, 1991; Yee, et al., 1990).

This paper integrates the pattern recognition and the explanatory heuristics that experts use into a program called Hoyle that learns to play two-person, perfect information, finite board games against an external expert. As in the schematic of Figure 1, whenever it is Hoyle's turn to move, a hierarchy of resource-limited

procedures called *Advisors* is provided with the current game state, the legal moves, and any useful knowledge (described below) already acquired about the game. There are 22 heuristic Advisors in two tiers. The first tier sequentially attempts to compute a decision based upon correct knowledge, shallow search, and simple inference, such as Victory's "make a move that wins the contest immediately." If no single decision is forthcoming, then the second tier collectively makes many less reliable recommendations based upon narrow viewpoints, like Material's "maximize the number of your markers and minimize the number of your opponent's." Based on the Advisors' responses, a simple arithmetic vote selects a move that is forwarded to the game-playing algorithm for execution.

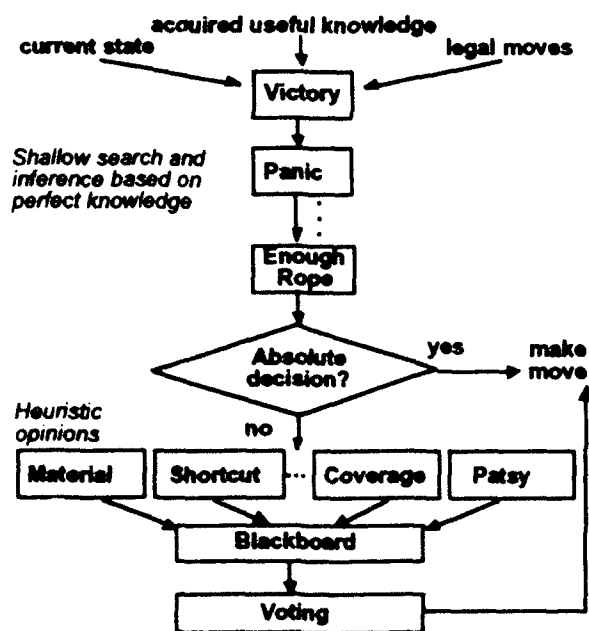


Figure 1: How Hoyle makes decisions.

The program learns from its experience to make better decisions based on acquired useful knowledge. *Useful knowledge* is expected to be relevant to future play and may be correct in the full context of the game tree.

Examples of useful knowledge include recommended openings and states from which a win is always achievable. Each item of useful knowledge is associated with at least one learning algorithm. The learning methods for useful knowledge vary, and include explanation-based learning, induction, and deduction. The learning algorithms are highly selective about what they retain, may generalize, and may choose to discard previously acquired knowledge. When individual Advisors apply current useful knowledge to construct their recommendations, they integrate these learning strategies. Full details on Hoyle are available in (Epstein, 1992).

Visual cues are integrated into Hoyle's decision-making process as new Advisors in the second tier. These Advisors react to lines and clusters of markers without reasoning. This is prompted by our observation that people guide their play with frequently-observed patterns of pieces before they understand their significance. The distinction drawn here between thinking and seeing in game playing is an important one. By "thinking" we mean the manipulation of symbolic data, such as "often-used opening gambit;" by "seeing" we mean inference-free, explanation-free reaction to visual stimuli. The three Advisors described here are directed toward the construction of a system that both uses and learns visual cues. They provide powerful performance gains and promise a natural integration with learning. This paper indicates how Hoyle, already a multistrategy learning program, can integrate knowledge about visual cues, and methods to learn them.

2. Using Predrawn Lines

Morris games have been played for centuries throughout the world on boards similar to

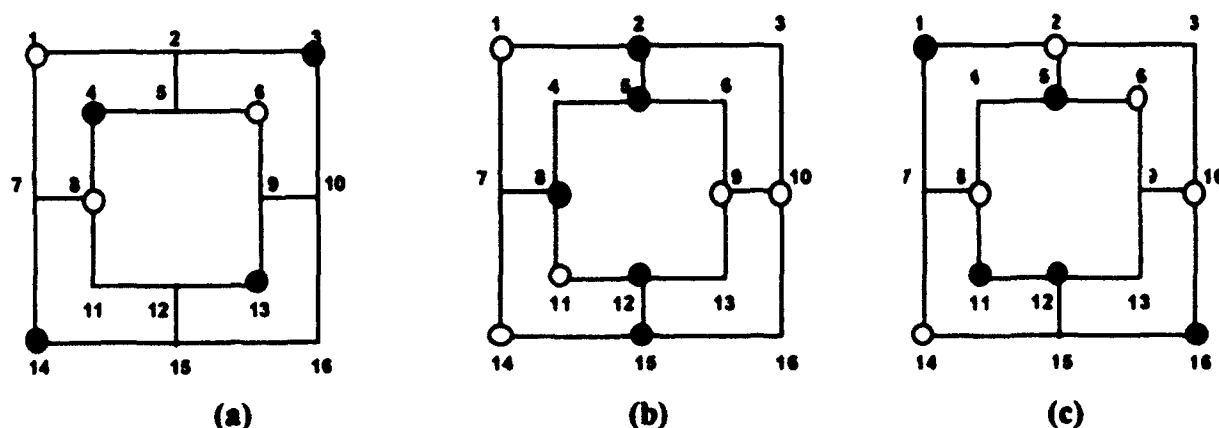


Figure 2: Some five men's morris states with white to move: (a) in the placing or the sliding stage, (b) and (c) in the sliding stage.

those in Figure 2. For clarity, we distinguish carefully here between a *game* (a board, markers, and a set of rules) and a *contest* (one complete experience at a game, from an initially empty board to some state where the rules terminate play). We refer to the predrawn straight lines visible in Figure 2 simply as *lines*. The intersection of two or more lines is a *position*. A position without a marker on it is said to be *empty*. Although the program draws pictures like those in Figure 2 for output, the internal, computational representation of any game board is a linear list of position values (e.g., black or white or blank) along with the identity of the mover and whether the contest is in the placing or sliding stage. The program also makes obvious representational transformations to and from a two-dimensional array to normalize computations for symmetry, but the array has no meaningful role in move selection. The game definition includes a list of predrawn lines and the positions on them.

A morris game has two contestants, black and white, each with an equal number of markers. A morris contest has two stages: a *placing stage*, where initially the board is empty, and the contestants alternate placing one of their markers on any empty position, and a *sliding*

stage, where a turn consists of sliding one's marker along any line drawn on the game board to an immediately adjacent empty position. A marker may not jump over another marker or be lifted from the board during a slide. Three markers of the same color on immediately adjacent positions on a line form a *mill*. Each time a contestant constructs a mill, she *captures* (removes) one of the other contestant's markers that is not in a mill. Only if the other contestant's markers are all in mills, does she capture one from a mill. (There are local variations that permit capture only during the sliding stage, permit hopping rather than sliding when a contestant is reduced to three near a contest's end, and so on.) The first contestant reduced to two markers, or unable to move, loses.

2.1 The Coverage algorithm

When a marker is placed on any position on a line, it is said to *affect* all the positions on that line, including its own. The *coverage* of a position is the multiset of all distinct positions that it affects. A marker positioned where two lines meet, induces two copies of its position. Thus the coverage of 3 in Figure 2(a) is {1, 2, 2-3, 10, 16}. A set of markers belonging to a single contestant P produces a *cover*, a multi-

set denoted $C_P = \{c_1 \cdot v_1, c_2 \cdot v_2, \dots, c_n \cdot v_n\}$ that lists the affected positions v_1, v_2, \dots, v_n and the number of lines c_i on which v_i lies that are affected by one of P 's markers. In Figure 2(a), the white cover is $C_W = \{2 \cdot 1, 2, 3, 2 \cdot 4, 5, 2 \cdot 6, 2 \cdot 7, 2 \cdot 8, 9, 11, 13, 14\}$. The *cover difference* $C \sim D$ for $C = \{c_1 \cdot v_1, c_2 \cdot v_2, \dots, c_n \cdot v_n\}$ and $D = \{d_1 \cdot w_1, d_2 \cdot w_2, \dots, d_m \cdot w_m\}$, is defined to be the multiset $C \sim D = \{x \cdot y \mid y = v_i \text{ for some } i = 1, 2, \dots, n; x \cdot y \in C; y \neq w_j \text{ for any } j = 1, 2, \dots, m\}$. In Figure 2(a), $C_B \sim C_W = \{10, 12, 15, 2 \cdot 16\}$ and $C_W \sim C_B = \emptyset$. We take the standard definitions from graph theory for adjacency, path, and path length.

A marker offensively offers the potential to group others along lines it lies on (*juxtaposition*) and to facilitate movement there (*mobility*), while it defensively obstructs the opposition's ability to do the same. The Coverage algorithm attempts to spread its markers over as many lines as possible, particularly lines already covered by the other contestant, and tries to do so on positions with maximal coverage. Assume, without loss of generality, that it is white's turn to move. In the placing stage, Coverage recommends a move to every empty position $c_i \cdot v_i \in C_B \sim C_W$ where $c_i > 1$. If there are no such positions, it recommends a move to every position in $C_B \sim C_W$ with maximal coverage. If there are no such positions of either kind, it recommends a move to every empty position with maximal coverage. In Figure 2(a) with White to move in the placing stage, $C_B \sim C_W = \{10, 12, 15, 2 \cdot 16\}$ so Coverage recommends a move to 16.

In the sliding stage, Coverage recommends each legal move that increases $|v_i|$, the number of the mover's distinct covered positions. Let (p, q) denote a sliding move from position p to position q . In Figure 2(b) the legal moves

$(1, 7)$, $(9, 6)$, $(9, 13)$, $(10, 3)$, $(10, 16)$, $(14, 7)$ change $|v_i|$ by $-1, +2, 0, 0, 0, -1$, respectively, so Coverage recommends $(9, 6)$. In the sliding stage, however, one's cover can also decrease. Therefore, Coverage also recommends each legal slide to a position $c_i \cdot v_i \in C_B$ where $c_i > 1$ but for which $c_i \leq 1$ in C_W . In Figure 2(c), where $C_B = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 6, 7, 8, 10, 3 \cdot 11, 3 \cdot 12, 2 \cdot 13, 2 \cdot 14, 2 \cdot 15, 2 \cdot 16\}$, $C_W = \{2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, 2 \cdot 5, 2 \cdot 6, 2 \cdot 7, 2 \cdot 8, 2 \cdot 9, 2 \cdot 10, 11, 13, 2 \cdot 14, 15, 2 \cdot 16\}$, and the legal moves are $(2, 3)$, $(6, 9)$, $(8, 4)$, $(8, 7)$, $(10, 3)$, $(10, 9)$, $(14, 7)$, $(14, 15)$, those vertices are 11, 12, 13, 15, so Coverage can only recommend $(14, 15)$.

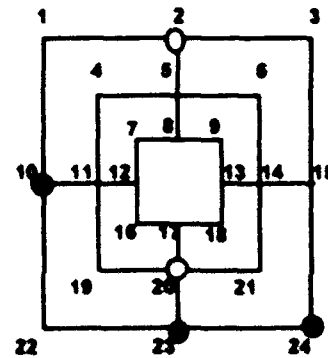


Figure 3: A placing state in nine men's morris, white to move.

2.2 The Shortcut algorithm

The Shortcut algorithm addresses long-range ability to move, and does so without forward search into the game graph. The algorithm for Shortcut begins by calculating the non-zero path lengths between pairs of same-color markers, including that from a marker to itself. For example, in Figure 3 the shortest paths between the white markers on 2 and 20 are $[2, 5, 6, 14, 21, 20]$, $[2, 3, 15, 14, 21, 20]$, and $[2, 5, 4, 11, 19, 20]$. Next, the algorithm selects those pairs for which the shortest non-zero length path between them is a minimum. It then retains only those shortest paths that meet the following criteria: every empty position

lies on some line without a marker of the opposite color, and at least one position on the path lies at the intersection of two such lines. All three paths identified for Figure 3 are retained because of positions 5, 14, and 5, respectively. Shortcut recommends a placing or sliding move to the middlemost point(s) of each such path. In Figure 3, Shortcut therefore recommends moves to the midpoints 6 and 14, 15 and 14, and 4 and 11. Computation for this algorithm, styled as spreading activation, is very fast.

2.3 Results with Coverage and Shortcut

Prior to Coverage, Hoyle never played five men's morris very well. There are approximately 9 million possible board positions in five men's morris, with an average branch factor of about 6. After 500 learning contests Hoyle was still losing roughly 85% of the time. Once Coverage was added, however, Hoyle's decisions improved markedly. (Shortcut was not part of this experiment; data averages results across five runs.) With Coverage, Hoyle played better faster; after 32.75 contests it had learned well enough to draw 10 in a row. The contests averaged 33 moves, so that the program was exposed during learning to at most 1070.5 different states, about .012% of the search space. From that experience, the program was judged to simulate expert play while explicitly retaining data on only about .006% of the states in the game graph.

In post-learning testing, Hoyle proved to be a reliable, if imperfect, expert at five men's morris. When the program played 20 additional contests against the model with learning turned off, it lost 2.25 of them. Thus Hoyle after learning is 88.75% reliable at five men's morris, still a strong performance after such limited experience and with such limited re-

tention in so large a search space. Additional testing displayed increasing prowess against decreasingly skilled opposition, an argument that expertise is indeed being simulated.

With a search space about 16,000 times larger than that of five men's, nine men's morris is a more strenuous test of Hoyle's ability to learn to play well. Because there is no definition of expert outcome for this game, we chose simply to let the program play 50 contests against the model. Without Coverage and Shortcut, Hoyle lost every contest. With them both, however, there was a dramatic improvement. Inspection showed that the program played as well as a human expert in the placing stage of the last 10 contests. During those 50 contests, which averaged 60 moves each, it lost 24 times, drew 17 times, and won nine times. (Some minor corrections to the model are now underway.) The first of those wins was on the 27th contest, and four of them were in the last six contests, suggesting that Hoyle was *learning* to play better. With the addition of less than 200 lines of game-independent code for the two new visually-cued Advisors, Hoyle was able to learn to outperform expert system code that was more than 11 times its length and restricted to a single game. The morris family includes versions for 6, 9, 11, and 12 men, with different predrawn lines. At this writing, Hoyle is learning them all quickly.

It should be noted that neither of these Advisors applies useful knowledge; instead, they direct the learning program's experience to the parts of the game graph where the key information lies, highly-selective knowledge that distinguishes an expert from a novice (Ericsson and Smith, 1991). If this knowledge is concisely located, as it appears to be in the morris games, and the learner can harness it, as Hoyle's learning algorithms do, the pro-

gram learns to play quickly and well. As detailed here, this general improvement comes at a mere fraction of the development time for a traditional game-specific expert system.

3. Learning Patterns

Hoyle is a limitedly rational system that deliberately avoids exhaustive search and complete storage of its experience. Consistent with this approach, the work described here retains only a small number of the patterns encountered during play, ones with strong empirical evidence of their significance. The program uses a heuristically-organized database to associate small geometrical arrangements of markers on the board with winning and losing. The associative, hierarchical pattern database is a new item of useful knowledge. The first level of the database contains states; the second level contains patterns.

The pattern database is constructed by the *pattern classifier*, an associated learning algorithm, as follows. At the end of each contest, every state that occurred during the contest is cached in a fixed-size hash table, noting the sequence number of the most recent contest in which it appeared and whether Hoyle won, lost, or drew there. Each new state in the pattern database is now matched against nine templates for a 3×3 grid, adjusted for symmetry and shown in Figure 4. A "?" in a template represents an X, an O, or an empty space; "#" is the don't care symbol. A *subpattern* is an instantiation of a template, e.g., X's in the corners of a diagonal. (Preliminary empirical tests showed this to be the smallest set of effective templates.)

The second level of the pattern database consists of those subpatterns which appear in at least two states of the first level. Most states

match several ways and therefore make multiple contributions to counting on the second level. Each subpattern also records the number of contests in which it participated in a win, a loss, and a draw. Thus a subpattern is a generalization over a class of states: those that have recently occurred with some frequency and contain simple configurations of pieces. Each subpattern is categorized as winning, drawing or losing based upon which kind of contest it appeared in most frequently.

? ? #	? ##	# ? #	? # ?	? ##
###	# ? #	# ? #	###	###
###	###	###	###	## ?
# ? #	???	? ##	# ? #	
###	###	# ? #	# ? #	
# ? #	###	## ?	# ? #	

Figure 4. The set of templates used by the pattern classifier.

It is important to forget in the pattern database, primarily to discount novice-like play during the early learning of a game. There will be winning contests, and patterns associated with them, that were due to the learner's early errors. We have therefore implemented two ways to forget in the pattern database. First, when a hash table for either states or patterns is full, and a new entry should be made, the least recently used entry is eliminated, based on its most recent contest number. Second, at the end of every contest, the number of times each state was encountered is multiplied by 0.9.

Patsy is an Advisor that ranks legal next moves based on their fit with the pattern database. Patsy looks at the set of possible next states resulting from the current legal moves. Each next state is compared with the subpattern level of the database. A matched

winning subpattern awards the state a +2, a matched drawing subpattern a +1, and a matched losing subpattern a -2. A state's score is the total of its subpattern values divided by the number of subpatterns in the cache. Patsy recommends the move whose next state has the highest such score. Ties are broken by random selection among the best moves.

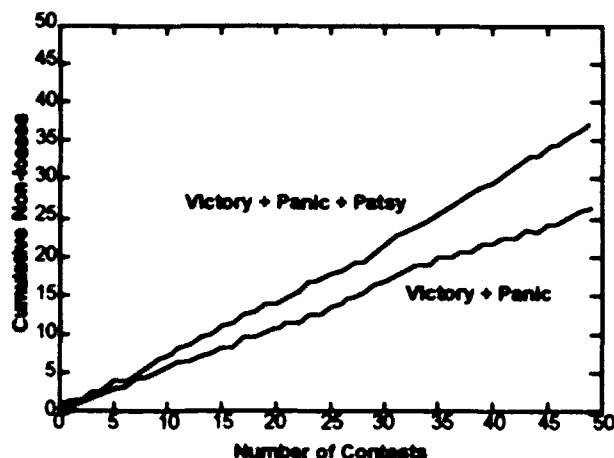


Figure 5. The performance of Hoylite with and without Patsy.

Patsy was tested within a severely pared-down version of Hoyle, called Hoylite here. Hoylite has only two of Hoyle's original Advisors, plus Patsy. The pattern classifier forms categories based on observed game states and associates responses to the observed states by learning during play. The hash table sizes were limited to 50 game states and 30 subpatterns. Three tournaments between Hoylite and a perfect tic-tac-toe player were run to assess the performance of Hoylite. The perfect player was a look-up table of correct moves. Each tournament was continued for 50 contests. The average cumulative number of Hoylite's wins and draws is plotted against contest number in Figure 5. The graph compares Hoylite's average performance against the perfect contestant with and without Patsy. Clearly Hoylite performs consistently better with Patsy.

There are many games that are played on a 3x3 grid. At this writing we are testing whether the same pattern templates in Figure 4 apply to several other games. We are also gradually adding Hoyle's Advisors to Hoylite, to see what conflicts, if any, arise. Finally, we are experimenting with more sophisticated pattern classifiers, ones that model the response of the human eye to arrangements such as lines of pieces and lines of open spaces.

4. Discussion

Predrawn game board lines are shown here to be important, readily accessible regularities that support better playing decisions. Historical data on patterns attractive to the human eye are demonstrably helpful in distinguishing good middlegame positions from mediocre ones. The brevity of the code required to capitalize on these visual cues for a variety of problems argues for the limitedly rational perspective of the architecture. The improvement the new Advisors have on play argues for the significance of visual representations as an integral part of decision making. When predrawn board lines are taken as visual cues for juxtaposition and mobility, Hoyle learns to play challenging games faster and better. Coverage and Shortcut in no way diminish the program's ability to learn and play the broad variety of games at which it had previously excelled (Epstein, 1992).

Our preliminary examination of the impact of a recognition-association competitive learning pattern classifier on several other expert knowledge sources and learning methods via a blackboard architecture is promising. The game played was a simple one, and only two of the 22 preexisting Advisors were included. A simple game was chosen to facilitate debugging the pattern classifier and measuring

performance against an absolute standard. More than two Advisors would have obscured the contribution of the pattern-associative component. Hoylite's pattern classifier is quite simple and does not learn new templates; it only learns which game states are important for the given set of templates. It can be seen from these preliminary results that a pattern recognition component can be smoothly integrated into a game playing system that involves reasoning and limited search.

Heuristic Advisors are needed most in the middlegame, where the large number of possible moves precludes search. It has been our experience with more complex games, where one would have many Advisors, that openings are typically memorized, and that the endgame can be well-played with Advisors that reason about known losing and winning positions. Inspection reveals that Shortcut and Coverage contribute to decisions only in the middlegame, while Patsy works on the opening and middlegame. In the full version of Hoyle, other Advisors cover the opening, and an experience-driven partial retrograde analysis learns enough useful knowledge to tune the endgame. In Hoylite, the other two Advisors, Victory and Panic, address the endgame, leaving Patsy to consider patterns important at the earlier stages. All three new Advisors prove to filter the middlegame alternatives to a few likely moves, ones that might then benefit from limited search.

Acknowledgments

J. G. was partially supported by a grant from the James S. McDonnell Foundation to the Human Information Processing Group at Princeton University. S. L. E. was partially supported by NSF Grant 9001936.

References

- Berliner, H. 1992. Pattern Recognition Interacting with Search. Tech. Rpt., CS-92-211, Carnegie Mellon University.
- Campbell, M. S. 1988. Chunking as an Abstraction Mechanism. Ph.D. diss., CMU.
- Epstein, S. L. 1992. Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems* 7: 547-586.
- Ericsson, K. A. and Smith, J. 1991. Prospects and Limits of the Empirical Study of Expertise. In *Toward a General Theory of Expertise*, ed. K. A. Ericsson and J. Smith. Cambridge: Cambridge University Press. 1-38.
- Fawcett, T. E. and Utgoff, P. E. 1991. A Hybrid Method for Feature Generation. In *Proc. 8th Int'l Workshop on Machine Learning*, 137-141. Morgan Kaufmann.
- Flann, N. S. 1992. Correct Abstraction in Counter-Planning: A Knowledge Compilation Approach. Ph.D. diss., Oregon State.
- Gelernter, H. 1963. Realization of a Geometry-Theorem Proving Machine. In *Computers and Thought*, ed. E. A. Feigenbaum and J. Feldman. NY: McGraw-Hill. 134-152.
- Holding, D. 1985. *The Psychology of Chess Skill*. Hillsdale, NJ: Lawrence Earlbaum.
- Levinson, R. and Snyder, R. 1991. Adaptive Pattern-Oriented Chess. In *Proc. 8th Int'l Machine Learning Workshop*, 85-89.
- Samuel, A. L. 1963. Some Studies in Machine Learning Using the Game of Checkers. In *Computers and Thought*, ed. E. A. Feigenbaum and J. Feldman. NY: McGraw-Hill. 71-105.
- Yee, R. C., Saxena, S., Utgoff, P. E. and Barto, A. G. 1990. Explaining Temporal Differences to Create Useful Concepts for Evaluating States. In *Proc. 8th Nat'l Conference on AI*, 882-888. AAAI Press.

Applying Multiple Learning Strategies for Classifying Public Health Data

John W. Sheppard
Department of Computer Science
The Johns Hopkins University
Baltimore, Maryland 21218
sheppard@cs.jhu.edu

Abstract

Analyzing data with the intent of inducing classification rules typically proceeds from a set of training data in which classifications are known. In the event classifications are unknown, algorithms exist for performing unsupervised learning to determine concept classes inherent in the data. In this paper, we describe experiments applying multiple learning strategies for classifying unlabeled data. Specifically, three unsupervised learning algorithms were applied to a large set of public health data in order to determine likely concept classes for the data based on the inherent features in the data. After inducing the concept classes, the data were processed by a decision tree algorithm in order to determine more efficient classification rules under the assumption that the concepts induced during unsupervised learning were correct.

Key words: Classification, unsupervised learning, clustering, decision trees

1. Introduction

The machine learning literature describes several approaches for classifying numerical

data. For example, decision trees (such as those generated by Quinlan's ID3 and C4 algorithms) select attributes as internal test nodes of a tree to determine the class to which a data point belongs, given at the leaves of the tree (Quinlan, 1986). Nearest neighbor algorithms store training examples paired with a classification (Aha *et al.*, 1991). When a new point is presented, the stored point that is *closest* in some sense (such as Euclidean distance or Hamming distance) is selected and the corresponding classification reported.

At times, labels providing classification information are not available with the training set. In these instances, unsupervised learning approaches may be employed to detect *clusters* of the data. These clusters can then be used to develop an initial set of classification labels (albeit non-symbolic) for the data.

In this paper, we will describe applying multiple learning strategies to a large set of psychiatric data (Eaton and Ritter, 1988; Eaton *et al.*, 1989). Specifically, we will compare three clustering algorithms and discuss the results of processing resultant clusters with a decision tree algorithm to provide an efficient classification strategy.

The psychiatric data, provided by the Johns Hopkins School of Public Health, consists of over 7,000 data points describing patients with respect to clinical depression or anxiety. Each data point has 58 fields indicating, for example, whether a patient has various fears, feelings of worthlessness, thoughts of suicide, etc. Our experiments used 20 binary attributes from the 58 provided. According to the School of Public Health, these 20 attributes characterize depression where the others provide demographic information and characterize anxiety. Note that none of the data, as of yet, have been classified (i.e., labels are not known *a priori*), thus motivating the analysis of unsupervised learning techniques.

The three clustering algorithms examined include a non-hierarchical approach, a hierarchical approach (thus resulting in a decision tree), and a connectionist approach. The nonhierarchical approach is based on a variation of MacQueen's *k*-means method (MacQueen, 1967). The standard *k*-means method assumes *k* clusters and fits the data in the clusters with the nearest centroids. The variation of this method used permits *k* to vary so that an estimate of the number of classes in the data may be determined.

The second cluster analysis approach is hierarchical. Hierarchical approaches either divide data or combine data in a tree structure. Divisive approaches begin with one large cluster and divide into smaller clusters based on the attributes. Agglomerative approaches begin with one cluster for each training sample and combine clusters based on similarity. The approach used in this part of the study is a divisive approach called *association analysis*. This approach selects an attribute to divide clusters by computing a matrix of chi-square coefficients for each attribute and selecting

the coefficient with the maximum sum of chi-square values (Everitt, 1974).

Finally, Rumelhart and Zipser (1986) describe a connectionist approach to clustering using competitive learning. The approach proceeds under the assumption that dominant attributes will generally determine the classification, and the network reinforces detection of the dominant attributes by strengthening weights associated with their corresponding input nodes. The output layer then applies a winner-take-all competition strategy to determine the cluster to which a data point belongs.

Since the experimental data used was not provided with classification labels, the second phase of the study consists of generating decision trees based on the classifications derived from the clustering techniques. Quinlan's C4 algorithm is applied to the results of all three clustering techniques, and the resulting trees compared to rules that can be derived from the clustering algorithms themselves.

2. Inducing Concept Classes Using Unsupervised Learning

There are many ways to characterize machine learning algorithms. One approach is based upon whether or not an external "teacher" exists. The two resulting types of learning algorithms are referred to as *supervised* learning and *unsupervised* learning. Typically, supervised learning proceeds when the results of some action are analyzed by a critic in comparison with known or expected results. Discrepancies between the two are used to determine ways to modify internal representations of the data so as to improve performance.

Unsupervised learning, on the other hand, does not have the advantage of an external teacher to determine "appropriate" behavior or "correct" classifications. Rather, data are examined and organized in such a way as to identify internal consistency. The class of cluster analysis algorithms generally fall within the set of unsupervised learning algorithms. In the following sections, we will describe the details of the three unsupervised learning algorithms used in this study.

2.1 Clustering by *k*-means

The first technique for clustering fits within the class of non-hierarchical techniques. Non-hierarchical clustering begins by selecting an initial set of clusters and alters the partitions so as to improve some metric. For example, *nearest centroid* methods attempt to develop partitions such that classification is made by comparing a point to the centroids of the clusters. The class corresponding to the nearest centroid is the one identified for that data point.

One of the most common approaches to non-hierarchical clustering is MacQueen's *k*-means algorithm (MacQueen, 1967). The *k*-means algorithm attempts to determine the *k* best clusters for a set of data such that classification is made by finding the cluster with the nearest Euclidean distance. Recall that the Euclidean distance between two points is computed as follows:

$$\text{dist}(p1, p2) = \sqrt{\sum_i (x1_i - x2_i)^2}$$

where $x1_i$ is the i th attribute of point $p1$ and $x2_i$ is the i th attribute of point $p2$. Since all of the attributes in the data set are binary, distance reduces to the square root of the Hamming distance.

The basic *k*-means algorithm consists of the following steps:

1. Select the first *k* data points as initial clusters with one member in each cluster.
2. Assign the remaining $m - k$ data points to the cluster with the nearest centroid.
3. After assigning each point, recompute the corresponding centroid of the cluster with the new point.
4. After all of the data points have been assigned, use the *k* clusters as seed points and pass through the data one more time for a final classification.

Variations of this algorithm exist in which the clusters converge to improved clusters. These variants require several passes through the data, but the law of diminishing returns may be experienced fairly early in the process.

Unfortunately, for our purposes, the basic *k*-means algorithm has a more serious drawback. This algorithm assumes the number of clusters is known and force fits all of the data into exactly *k* clusters. For this reason, MacQueen also proposed a variant in which the number of clusters is not known. This algorithm is the one selected for this study and is composed of the following steps:

1. Select values for an initial *k* and two additional parameters, *C* (coarsening) and *R* (refining).
2. As in the basic *k*-means algorithm, select the first *k* data points as the initial clusters.
3. Compute all of the pairwise distances between each of the cluster centroids. If the smallest distance is

less than C , then merge the two corresponding clusters and recompute the corresponding centroid. Continue merging until no other merges occur.

4. Assign the remaining $m - k$ data points one at a time to the cluster with the nearest centroid. If the distance to the nearest centroid is greater than R , then consider the point a new cluster and goto step 3.
5. After all of the data points have been assigned, use the cluster centroids as seed points and pass through the data one last time assigning the points to the clusters with the nearest centroids.

This algorithm can also follow convergent approaches, and as before, it has been found that diminishing returns exhibit themselves early in the process.

2.2 An associative clustering algorithm

For the second clustering technique, a hierarchical approach was used. Hierarchical clustering produces a decision tree by which data points can be classified according to the determined clusters. In general, hierarchical clustering is either *divisive* or *agglomerative*. Agglomerative approaches proceed with each data point treated as individual clusters. Clusters are then combined to form higher level clusters. This process continues until a group of high level clusters (or a single cluster) is identified. Divisive approaches begin with a single cluster and divide the cluster into sub-clusters. This process continues recursively until base clusters are determined.

In addition, hierarchical approaches can be classified as *monothetic* or *polythetic*. Monothetic techniques attempt to cluster

according to single attributes where polythetic techniques cluster according to the values of all of the attributes.

The technique used in this part of the study is a monothetic, divisive cluster analysis algorithm called *association analysis* (Everitt, 1974). Association analysis divides clusters by selecting the single attribute that provides the "best" split. The concept of a best split has been defined in several ways. For example, decision tree algorithms frequently employ concepts from Shannon's information theory to select the attribute that provides the most information independent of the actual values of the attributes (Shannon, 1948).

Association analysis selects attributes that maximize the chi-square coefficients of the data. Recall that chi-squared is computed as follows:

$$\chi^2 = \frac{(n - 1) s^2}{\sigma^2}$$

where s^2 is that sample variance, σ^2 is the population variance, and n is the sample size.

For association analysis, we assume all of the attributes are binary. The computation of the chi-square coefficients on binary data is similar to the standard equation. Let $attrib_{ij}$ be the j th attribute of the i th data point and $attrib_{ik}$ be the k th attribute of the i th data point. Then

$$a_{jk} = \sum_i attrib_{ij} * attrib_{ik}$$

$$b_{jk} = \sum_i (1 - attrib_{ij}) * attrib_{ik}$$

$$c_{jk} = \sum_{vi} \text{attrib}_{vj} * (1 - \text{attrib}_{vi})$$

$$d_{jk} = \sum_{vi} (1 - \text{attrib}_{vj}) * (1 - \text{attrib}_{vi})$$

Then the chi-square coefficients are simply computed as

$$\chi^2_{jk} = \frac{(ad - bc)^2 n}{(a + b)(a + c)(b + d)(c + d)}$$

and the attribute is selected such that $\left\{ \sum_{j=k} \chi^2_{jk} \right\}$ is maximized.

2.3 Clustering by competitive learning

For the final clustering technique, a connectionist algorithm was selected. In particular, the competitive learning neural network described by Rumelhart and Zipser (1986) was implemented. (Note that variants on this network are described by von der Malsburg (1973) and Grossberg (1987)) The idea behind competitive learning is that the network develops a set of "feature detectors." When data containing a learned feature are submitted to the network, then the activity of the network identifies which feature is present. To identify features, nodes within the network "compete" among themselves to respond to the stimulus pattern. The node that wins the competition has the feature associated with it. Consequently, when that node becomes active, the feature has been identified.

In order to train a competitive learning network, the weight matrix is constructed with m rows and n columns, where m = the number of output nodes and n = the

number of input nodes. The weight matrix is initialized with the following:

$$w_{ij} = \frac{1}{n_o} \pm \delta; \quad \forall i, j$$

where n_o is the number of nodes at the input layer and δ is a small random number generated for each weight.

The network is trained by processing a set of training data. Then, for each output node in the network, and for each training case, a "winning" node is determined. This winner is used to determine which node's weights are to be updated. The winner is determined as follows:

$$\text{winner} = \max_j \left\{ \sum_{i=0}^{n_i-1} w_{ji} I_i \right\}$$

where w_{ji} is the value in the weight matrix corresponding to row j and column i , I_i is the activation value of input node i , and j ranges over the number of outputs.

The competitive learning rule is then applied to the winner for the given training instance. In other words, the weights in the weight matrix are only modified for the connections between the input nodes and the winning output node. The update rule for modifying the weights in the network is as follows:

$$\Delta w_{ji} = \eta \left\{ \frac{I_i}{\left\{ \sum_{k=1}^{n_i} I_k \right\}} - w_{ji} \right\}$$

where Δw_k is the change in the weight matrix and η is a learning factor.

The clusters are identified by winning nodes when a data point is presented to the network. A further analysis of the network can help to identify the attributes that are most significant in clustering the data. In particular, since the update rule "strengthens" the connections between winning nodes and the significant inputs (i.e., attributes), the strong attributes for a given class will have weights greater than $1/n_c$.

3. Inducing Decision Trees

The three clustering algorithms described in the previous sections provide approaches to labeling data not previously labeled for classification. Once labels have been assigned, the next step is to determine efficient and effective means for classifying data according to the concepts learned that have not previously been encountered. One approach for such concept learning is the induction of decision trees. Perhaps the most famous decision tree algorithm is ID3 and its successor C4, both developed by Quinlan (1986).

ID3 and C4 allow attributes to be multi-valued (i.e., they do not limit attributes to binary values) and construct classification trees by selecting attributes that provide the best split among the data according to known classifications. The resulting tree is then used to classify data including data not used in training. The rules generated for the decision tree then permit classification to generalize so as to classify new data. Of course, since we do not know what the correct classifications are for our experiments, it is difficult to determine how well the trees generalize. (Note that C4, the program used in these experiments,

automatically constructs trees on a subset of the training data using ten-way cross-validation and selects a tree that generalizes the best on the remaining data.)

In order for ID3 and C4 to determine the best attribute at a given point, Quinlan incorporated the information entropy function described by Shannon (1948). The information value of a set of data T is

$$I(T) = - \sum_{i=1}^{|C|} \frac{\text{freq}(c_i, T)}{|T|} \log_2 \frac{\text{freq}(c_i, T)}{|T|}$$

where C is the set of classes, T is the set of training instances, and $\text{freq}(c_i, T)$ is the frequency of class i occurring in T . The expected information value of attrib_j is

$$E(\text{attrib}_j) = \sum_{i=1}^{v_j} \frac{|T_i|}{|T|} I(T_i)$$

where v_j is the number of values attrib_j can have and T_i is the subset of T with attrib_j having the i^{th} value. Then the information gain is simply $I(T) - E(\text{attrib}_j)$. The attribute with the maximum gain is selected for the root of the current subtree. C4 adds several techniques for pruning the trees, thus making the final trees more efficient than the initial ones (Quinlan, 1987). Also, C4 applies a gain ratio criterion for its splitting criterion, but when all attributes are binary, the result is identical to applying information gain.

4. The Public Health Data

For this study, psychiatric data on anxiety and depression were analyzed. This data set consisted of over 7,000 samples with 58 binary attributes. The data set was

collected from the East Baltimore Epidemiologic Catchment Area (ECA) Program and was supplied by the Johns Hopkins School of Public Health (Eaton and Ritter, 1988; Eaton *et al.*, 1989). The data was not categorized prior to analysis, so the object of the study was to identify regularities within the data that might suggest natural classifications.

For this study, the data set was reduced in three ways. First, several of the samples had attributes with unknown values. All samples with more than five unknown attributes were eliminated from the data set. Second, since all of the attributes were negative characteristics, all samples in which all of the attributes were zero were also removed. This resulted in a data set of approximately 2,000 points. Finally, 20 binary attributes were identified as specifically relevant to depression. Therefore, all of the clustering algorithms limited consideration to these 20 attributes. The 20 attributes used in the study are as shown in Table I:

5. Experiments

As mentioned above, the experiments described in this report followed four major steps. First, *k*-means clustering (with the described modification) was applied. Second, the reduced data set was processed by association analysis to generate a decision tree. Third, the competitive learning neural network was applied to data. Finally, classification labels were assigned to the data points based on the results for each of the clustering methods and decision trees were generated by C4. The results of C4 generating decision trees will be discussed at the end of each relevant section. Unfortunately, space limitations prevent us from including all of these trees. The

following sections describe the results of the clustering studies.

5.1 *K*-means clustering

K-means clustering provides a technique for determining clusters within the data using a principle based on nearest neighbor. As such, it is not capable of handling overlapping clusters. On the other hand, it is capable of clustering based on all of the attributes rather than limiting its view to single clusters (i.e., it is polythetic). Of course, this makes it more difficult to determine relevant rules for classification, but we attempt to extract rules from the results of the analysis.

Recall that this technique requires an initial value for *k* to be provided as well as a coarsening and refining parameter. The latter two parameters were determined empirically, and *k* was set initially to 10. In particular, the coarsening parameter was set to 0.5 and the refining parameter was set to 1.95. It was found that coarsening was highly sensitive to values near 1.0 and refining was highly sensitive near 2.0. *K*-means was actually applied last, so the parameters were selected to yield results similar to the other two techniques.

Following *k*-means clustering on the public health data, 12 clusters were identified. Attributes of their centroids are listed in Table II. It was found that the two least similar clusters were Cluster 8 and Cluster 11. It is believed that these clusters would represent the extremes on the spectrum of depression. As such, it would be valuable to decipher the centroids to determine the relevant characteristics. Cluster 8 showed very low incidence of depression related attributes with the exception of increased eating. On the other hand, Cluster 11 show high incidence of depression related

Table I. Attribute for Public Health Data on Depression and Anxiety

1.	CONCENT	Trouble concentrating
2.	CRYING	Crying spells
3.	DEATHT	Thought about death
4.	DEATHW	Wanted to die
5.	EATLESS	Lost appetite
6.	GAIN2LB	Eating increased
7.	HOPELESS	Life hopeless
8.	LOSE2LB	Lost weight
9.	MOVMORE	Moving all of the time
10.	SAD2WK	Sad for two weeks
11.	SAD2YRS	Sad for two years
12.	SEXDIM	Diminished interest in sex
13.	SLPLESS	Trouble falling asleep
14.	SLPMORE	Sleeping too much
15.	SUIDTRY	Attempted suicide
16.	SUTHINK	Thought of suicide
17.	THINKSLO	Thoughts slower
18.	TIRED	Tired out
19.	TMSLOW	Talked more slowly
20.	WSG2WK	Worthless, sinful, guilty

attributes in all but two attributes—increased eating and moving all the time.

The attributes at the centroids can be considered as weighted “presence” of that attribute in determining whether or not a point belongs to some cluster. These weights spanned 0.1 to 0.9, so a decision tree generated by C4 will not divide cleanly along the attributes (as one might expect from a hierarchical clustering analysis such as the one discussed in the next section). In fact the pruned decision tree generated by C4 has 62 paths and a maximum depth of 13 steps.

It is interesting to note that the top attributes of the C4 tree are feelings of worthlessness, being sad for two weeks, and thinking slowly. The first two were also

found to be significant in the study reported in Eaton and Ritter (1988). On the other hand, thoughts of death (considered to be the most significant attribute in the Eaton, *et al.* study) appears fairly deep in the tree.

5.2 Chi-square clustering

The results of running the chi-square association analysis on the public health data was a decision tree that yielded 16 classifications (Table III). Since the basic goal in classifying this data is to determine whether or not a patient is depressed, it is apparent that subcategories may exist within the data. Unfortunately, we are not in a position to determine the nature of these subcategories without the basic labeling of the data.

Table II. Cluster Attributes from K-Means Algorithm.

<u>CLUSTER</u>	<u>HIGHEST ATTRIBUTES</u>	<u>LOWEST ATTRIBUTES</u>
1	TIRED	SUIDTRY
2	EATLESS	CONCENT, CRYING, DEATHW, GAIN2LB, HOPELESS, MOVMORE, SAD2WK, SEXDIM, SUIDTRY, SUTTHINK, THINKSLO, WSG2WK
3	WSG2WK	EATLESS, GAIN2LB, LOSE2LB, SEXDIM, SUIDTRY, SUTTHINK, THINKSLO
4	HOPELESS	EATLESS, GAIN2LB, LOSE2LB, SAD2WK, SAD2YRS, SLPMORE, SUIDTRY, THINKSLO, TMSLOW
5	TIRED	CRYING, DEATHT, DEATHW, GAIN2LB, HOPELESS, MOVMORE, SAD2WK, SAD2YRS, SLPLESS, SUIDTRY, SUTTHINK, WSG2WK
6	SLPLESS	CONCENT, DEATHT, DEATHW, EATLESS, GAIN2LB, HOPELESS, LOSE2LB, SAD2WK, SAD2YRS, SEXDIM, SLPMORE, SLPLESS, SUIDTRY, SUTTHINK, TMSLOW WSG2WKS
7	DEATHT	CONCENT, CRYING, DEATHW, EATLESS, GAIN2LB, HOPELESS, LOSE2LB, SAD2WK, SAD2YRS, SEXDIM, SLPMORE, SLPLESS, SUIDTRY, SUTTHINK, TMSLOW WSG2WKS
8	SAD2WK	DEATHW, GAIN2LB, MOVMORE, SLPMORE, SUIDTRY, SUTTHINK, TMSLOW, WSG2WK
9	GAIN2LB	CONCENT, CRYING, DEATHW, EATLESS, HOPELESS, LOSE2LB, SAD2YRS, SUIDTRY, SUTTHINK, THINKSLO, TMSLOW, WSG2WK
10	CONCENT, THINKSLO	MOVMORE, SUIDTRY, SUTTHINK
11	DEATHT, DEATHW, HOPELESS, LOSE2LB, MOVMORE, SAD2WK, SUTTHINK, TIRED, WSG2WK	EATLESS, GAIN2LB, SLPLESS SLPMORE, THINKSLO, TMSLOW
12	CONCENT, DEATHT, DEATHW, HOPELESS, SAD2WK, SUTTHINK, THINKSLO	GAIN2LB, LOSE2LB

Table III. Decision Rules from Chi-Square Clustering.

CLUSTER	RULE
1	DEATHW=1, CONCENT=1, SUTTHINK=1, HOPELESS=1
2	DEATHW=1, CONCENT=1, SUTTHINK=1, HOPELESS=0
3	DEATHW=1, CONCENT=1, SUTTHINK=0, THINKSLO=1
4	DEATHW=1, CONCENT=1, SUTTHINK=0, THINKSLO=0
5	DEATHW=1, CONCENT=0, SUIDTRY=1
6	DEATHW=1, CONCENT=0, SUIDTRY=0, LOSE2LB=1
7	DEATHW=1, CONCENT=0, SUIDTRY=0, LOSE2LB=0, SAD2WK=1
8	DEATHW=1, CONCENT=0, SUIDTRY=0, LOSE2LB=0, SAD2WK=0
9	DEATHW=0, SLPMORE=1
10	DEATHW=0, SLPMORE=0, SUTTHINK=1
11	DEATHW=0, SLPMORE=0, SUTTHINK=0, CONCENT=1, DEATHT=1
12	DEATHW=0, SLPMORE=0, SUTTHINK=0, CONCENT=1, DEATHT=0
13	DEATHW=0, SLPMORE=0, SUTTHINK=0, CONCENT=0, SAD2WK=1, HOPELESS=1
14	DEATHW=0, SLPMORE=0, SUTTHINK=0, CONCENT=0, SAD2WK=1, HOPELESS=0
15	DEATHW=0, SLPMORE=0, SUTTHINK=0, CONCENT=0, SAD2WK=0, THINKSLO=1
16	DEATHW=, SLPMORE=0, SUTTHINK=0, CONCENT=0, SAD2WK=0, THINKSLO=0

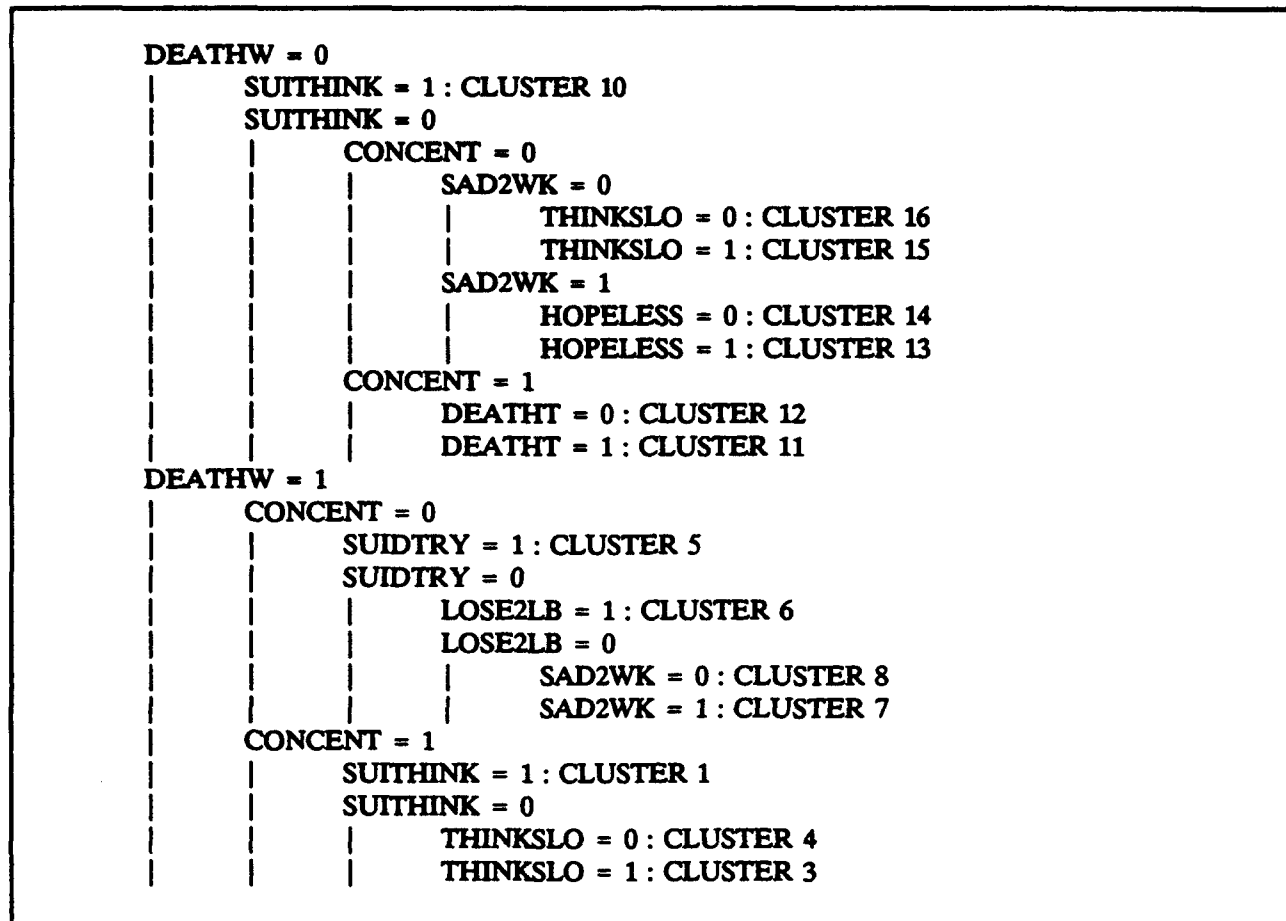
Perhaps the most interesting observation to be made from this analysis was determining which of the attributes are considered most significant in separating the data. Since association analysis is a hierarchical technique, attributes used near the root of the tree differentiate between high level clusters where attributes used near the leaves of the tree differentiate between finer grained clusters. So the first observation is that the attribute DEATHW (i.e., wanting to die) should be highly indicative of whether or not a patient is depressed, assuming only the two classifications exist and a single attribute can distinguish the two clusters. Of course, this assumption may be totally inappropriate. Another plausible interpretation is that the clusters generated by this technique (and by the others) represent "degrees" of depression. As such, wanting to die may suggest more severe depression while the lack of such

thoughts may not completely eliminate depression.

It is also interesting to note that, in Eaton and Ritter (1988), classification according to dysphoria (i.e., general depression) indicates the highest correlation with thoughts of death. Further, two of the four classes of depression identified indicated dysphoric symptoms (indicated by thoughts of death) as a leading attribute. The attribute, SAD2WKS was also considered highly indicative of dysphoria. The chi-square analysis performed here also found this attribute to be significant but nearer the leaves of the tree.

Finally, the results of the chi-square approach were processed by C4 (Table IV). The most important observation that we made from the resulting tree was that the two trees are very similar but not identical. One would expect the trees to be similar since the classifications were initially made

Table IV. C4 Decision Tree from Chi-Square Clustering.



with attributes providing "perfect" splits of the data. The reason for the difference in the trees lies in the metric used to select an attribute. In the chi-square approach, the chi-square metric is used to find high level variation along the lines of the attributes. C4 attempts to select attributes to build the decision tree under a similar motivation, but the metric used is information gain. The information gain metric attempts to evenly split the data into near equal subsets. In fact, we find that the maximum depth of the chi-square tree is six and the maximum depth of the C4 tree is five. For 16 classes, optimal depth of the tree (assuming equal sized clusters) would be four on each branch. No calculations were conducted to determine expected cost to classify based on

the size of the data set and the path lengths; however, it is conjectured that C4's tree will be slightly better.

5.3 Competitive clustering

Finally, the competitive learning algorithm was applied to the public health data. This approach assumes there will be fewer clusters than attributes, so since there were 20 attributes, one naturally expects fewer than 20 clusters. Indeed, competitive learning identified twelve clusters as in *k*-means. The results of applying competitive learning are shown in Table V.

One should observe right away that the results are very similar to the *k*-means

Table V. Cluster Attributes from Competitive Learning Algorithm.

<u>CLUSTER</u>	<u>HIGHEST ATTRIBUTES</u>	<u>LOWEST ATTRIBUTES</u>
1	CONCENT, THINKSLO, TMSLOW	HOPELESS, SUIDTRY, SUTHINK
2	DEATHW, SUTHINK	CRYING, EATLESS, SLPLESS, SLPMORE
3	DEATHT	SUIDTRY
4	HOPELESS	LOSE2LB
5	GAIN2LB, SLPMORE	CONCENT, CRYING, DEATHW, EATLESS, SAD2YRS, SUIDTRY
6	EATLESS, LOSE2LB	DEATHW, EATLESS, SUIDTRY
7	SEXDIM	DEATHW, HOPELESS, LOSE2LB, SAD2YRS, SUIDTRY, SUTHINK, WSG2WK
8	SAD2WK, WSG2WK	SUIDTRY, SUTHINK
9	CRYING, HOPELESS, SAD2WK	SUIDTRY
10	TIRED	GAIN2LB, SAD2WK, SUIDTRY, SUTHINK
11	MOVMORE	SUIDTRY
12	SLPLESS, TIRED	HOPELESS, LOSE2LB, SUIDTRY, SUTHINK

results. First, the number of clusters is the same. Examining the attributes that are significant (by examining the values of the weight matrix) reveals that there are several similar clusters within the network, and some of these clusters correspond to the *k*-means clusters. However, it also appears that the *k*-means clusters are more distinct. One possible explanation for this is that the competitive learning algorithm has difficulty due to its sensitivity to the order in which the data are presented.

The corresponding decision tree generated by C4 is also very complex. It has 54 paths and a maximum depth of 17, thus its complexity is analogous to the *k*-means tree. One significant difference, however, is the selection of primary attributes (i.e., attributes near the root). The clusters generated from competitive learning

resulted in primary attributes of sleeplessness, crying, and hopelessness. Only the latter is one of the significant attributes in Eaton and Ritter (1988). In fact, the more significant attributes appeared nearer to the leaves in this tree.

6. Discussion

The results of this study suggest that several degrees of clinical depression may exist. This is evident by the fact that all three clustering algorithms identified on the order of 12 to 16 clusters within the data. Recall that this data was reduced so as to consider attributes most relevant to depression; however, some carryover from anxiety is expected to have occurred. Nevertheless, the number of clusters identified is strong evidence that finer classifications may exist for depression.

In a previous study applying latent class analysis (Grove *et al.*, 1987), a reduced set of clusters was assumed. Specifically, this study assumed two classes. The studies reported in (Eaton and Ritter, 1988; Eaton *et al.*, 1989) also applied latent class analysis and found four classes. In a more recent study (Furukawa and Sumita, 1992), a hierarchical clustering algorithm was applied to a similar data set and three clusters identified. Unfortunately, the data set used was extremely small (40 subjects) thus making it difficult to compare with our results.

For our experiments, we were able to observe the following. First, both *k*-means and competitive learning found 12 clusters with similar attributes. Unfortunately, the "significance" of the attributes for the two techniques (as evidenced by the C4 decision trees) did not agree. Second, the association analysis generated 16 clusters by considering clean partitions of the data along individual attributes. Now it is unreasonable to assume that all 20 of the attributes are independent, so the idea that such a clean partitioning can occur becomes difficult to accept. In fact, many of the classification rules have combinations of thoughts of death, wanting to die, thinking about suicide, and attempting suicide. But the other rules seem to suggest grades of depression when combinations of these attributes (and others) have conflicting values (e.g., Cluster 8 included wanting to die but thinking about suicide was notably absent).

Note that both *k*-means and competitive learning are polythetic algorithms while chi-square clustering and C4 are monothetic algorithms. From this it should not be surprising that chi-square clustering and C4 yield comparable results as do *k*-means and competitive learning. It is also

understandable, given this difference, that the C4 trees for *k*-means and competitive learning would be much more complex than the C4 tree for chi-square clustering.

Aside from the obvious differences in the trees generated by all three techniques, these trees also had several similarities. First, the principal attributes all tended to agree with Eaton and Ritter (1988) and the trees tended to be highly complex. Further, in post-pruning, all three trees showed minimal rearrangement. Thus the initial trees appeared to be near optimal for the training set.

Several additional analyses could be done on this data. First, if the data were classified, then the classifications could be compared to the clusters identified to determine if, indeed, degrees or hierarchies of depression exist. Second, closer examination of the centroids of the clusters generated by *all three* techniques may be useful in determining how similar the tree results really are. For example, it is possible that the 12 clusters identified by *k*-means may closely correlate to the 12 clusters identified by competitive learning (although the decision trees seem to indicate the opposite). Unfortunately, time did not permit such a correlation analysis to be run.

Finally, additional classification algorithms could provide interesting results. For example, AutoClass by Cheeseman *et al.* (1988) is a Bayesian classification tool that attempts to identify the most probable set of clusters within the data. Running AutoClass on the data would provide another valuable data point in determining the character of the depression data.

Another alternative clustering system that we may apply is Fisher's COBWEB (1987).

COBWEB is an incremental system for hierarchical conceptual clustering. While our problem does not need to be examined incrementally, COBWEB offers the advantage of applying a different utility measure (i.e., category utility) to evaluate generated clusters. It also constructs the classification tree by using traditional search operators such as merging and splitting (corresponding to generalization and specialization respectively). Finally, since it represents concepts probabilistically, COBWEB should be better suited to the large data set than more rigid clustering algorithms such as *k*-means or chi-square clustering.

Traditional conceptual clustering as introduced by Michalski and Stepp (1983) and further developed by Stepp and Michalski (1986) rely on incorporating background knowledge in evaluating the quality of the resulting clusters. In our problem, little to no background knowledge was available, so this traditional approach could not be applied easily. COBWEB's advantage over CLUSTER/2 (Michalski and Stepp, 1983) or CLUSTER/S (Stepp and Michalski, 1986) is that the evaluation function is domain independent. However, we would expect the availability of domain knowledge to improve classification strategies.

7. Summary

In this paper we presented the results of three approaches to analyzing and clustering a large set of psychiatric data. As a result of this study, it is apparent that depression cannot be categorized either as simply present or absent. Further, it is unlikely as few as three or four classes of depression are sufficient. The results of this study suggest that there are many degrees of depression ranging from no depression to

severe depression. Further, depending on the means by which clusters are identified, it is also apparent that a relatively well defined (although not necessarily small) set of rules can be derived to assist in classifying a patient as fitting in one of the categories. These rules may be expressed either in terms of a decision tree (as in association analysis) or as a linear equation (as in the neural net). And in each of these cases, additional decision trees can be constructed which clearly delineate the rules to be applied for classification.

Acknowledgements

I would like to thank Joe Gallo of the Johns Hopkins School of Public Health for providing the patient data and for his assistance in interpreting the data. I would also like to thank Steven Salzberg and Simon Kasif for their ideas and support through this research. Finally, I would like to thank David Aha for providing several insightful comments after reading an early version of the paper. Support for this research was provided by ARINC Research Corporation.

References

- Aha, D. W., D. Kibler, and M. K. Albert, "Instance Based Learning Algorithms," *Machine Learning*, Vol. 6, pp. 37-66, 1991.
- Cheeseman, Peter, James Kelley, Matthew Self, John Stutz, Will Taylor, and Don Freeman, "AutoClass: A Bayesian Classification System," *Proceedings of the Fifth International Workshop on Machine Learning*, San Mateo, California: Morgan Kaufmann, 1988.
- Eaton, W. W., and C. Ritter, "Distinguishing Anxiety and Depression with Field Survey

Data," *Psychological Medicine*, Vol. 18, pp. 155-166, 1988.

Eaton, W. W., A. McCutcheon, A. Dryman, and A. Sorenson, "Latent Class Analysis of Anxiety and Depression," *Sociological Methods and Research*, Vol. 18, No. 1, pp. 104-125, 1989

Everitt, B., *Cluster Analysis*, Wiley and Sons, New York, 1974

Fisher, D. H., "Knowledge Acquisition Via Incremental Conceptual Clustering," *Machine Learning*, Vol. 2, pp. 139-172, 1987.

Furukawa, T., and Y. Sumita, "A Cluster-Analytically Derived Subtyping of Chronic Affective Disorders," *Acta Psychiatr Scand*, Vol. 85, pp. 177-182, 1992.

Grossberg, S., "Adaptive Pattern Classification and Universal Recoding: Part I. Parallel Development and Coding of Neural Feature Detectors," *Biological Cybernetics*, Vol. 23, pp. 121-134, 1976.

Grossberg, S., "Competitive Learning: From Interactive Activation to Adaptive Resonance," *Cognitive Science*, The Cognitive Science Society, Vol. 11, pp. 23-63, 1987.

Grove, W. M., N. C. Andreasen, M. Yound, J. Endicott, M. B. Keller, R. M. A. Hirschfeld, and T. Reich, "Isolation and Characterization of a Nuclear Depressive Syndrome," *Psychological Medicine*, Vol. 17, pp. 471-484, 1987.

MacQueen, J. B., "Some Methods for Classification and Analysis of Multivariate Observations," *Proceedings of the Symposium on Mathematical Statistics and Probability*, University of California Press, Berkeley, California, pp. 281-297, 1967.

Michalski, R. S., and R. E. Stepp, "Learning from Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach*, Vol. 1, eds. R. Michalski, J. Carbonnel, and T. Mitchel, Tioga Publishing Company, Palo Alto, California, pp. 331-363, 1983.

Quinlan, J. R., "Induction of Decision Trees," *Machine Learning*, Vol. 1, pp. 81-106, 1986.

Quinlan, J. R., "Simplifying Decision Trees," *Journal of Man-Machine Studies*, Vol. 27, pp. 221-234, 1987.

Rumelhart, D. E. and D. Zipser, "Feature Discovery by Competitive Learning," in *Parallel Distribute Processing*, David E. Rumelhart and James L. McClelland (eds.), The MIT Press, Cambridge, Massachusetts, pp. 151-193, 1987.

Shannon, C. E., "A Mathematical Theory of Communications," *Bell System Technical Journal*, Vol. 27, pp. 379-423, 1948.

Stepp, R. E., and R. S. Michalski, "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects," in *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, eds. R. Michalski, J. Carbonnel, and T. Mitchel, Morgan Kaufmann Publishers, Palo Alto, California, pp. 471-498, 1986.

von der Malsburg, C., "Self-organizing of Orientation Sensitive Cells in the Striate Cortex," *Kybernetik*, Vol. 14, pp. 85-100, 1973.

Author Index

Altmann, Erik M.	50	Kasif, Simon	138
Arcos, Josep Lluís	42	Malerba, Donato	276
Baffes, Paul T.	69	Michalski, Ryszard S.	3, 188
Bleken, Erlend	92	Mooney, Raymond J.	69
Bloedorn, Eric	188	Pazzani, Michael J.	276
Bruynooghe, Maurice	92	Plaza, Enric	42
Chan, Philip K.	150	Ram, Ashwin	259
Coget, Vincent	92	Saitta, Lorenza	234
Craven, Mark W.	207	Salzberg, Steven	138
De Garis, Hugo	250	Santamaria, Juan Carlos	259
De Raedt, Luc	92	Segen, Jakub	293
Drobnic, Matija	31	Semeraro, Giovanni	276
Duff, David	76	Shavlik, Jude W.	207
Epstein, Susan L.	301	Sheppard, John W.	309
Esposito, Floriana	276	Sirag Jr., David J.	166
Gams, Matjaz	31	Sleeman, Derek	107
Gelfand, Jack	301	Sommer, Edgar	180
Ghil, Chaouat	92	Stolfo, Salvatore J.	150
Giordana, Attilio	234	Subramanian, Devika	218
Gordon, Diana F.	218	Swennen, Bart	92
Graner, Nicolas	107	Tecuci, Gheorghe	76
Heath, David	138	VanLehn, Kurt	19
Hieb, Michael R.	3	Whitehall, Bradley L.	166
Jenkins, Wayne T.	58	Widmer, Gerhard	123
Jones, Randolph M.	19	Wnek, Janusz	188